MASTER EN SCIENCES MATHÉMATIQUES

Mesure de densité en analyse des données : algorithmes, programmation C et comparaison de méthodes

BOURGEOIS, Emmanuel

Award date:
1992

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR
FACULTE DES SCIENCES
Année académique : 1991 - 1992

MESURE DE DENSITE EN ANALYSE DES DONNEES : ALGORITHMES, PROGRAMMATION C ET COMPARAISON DE METHODES.

Emmanuel BOURGEOIS

Promoteur : Professeur J.-P. RASSON

André HARDY
Département de Mathématique
Facultés Universitaires de Namur
Rempart de la Vierge 8
B - 5000 NAMUR - BELGIUM

Facultés Universitaires Notre-Dame de la Paix
FACULTE DES SCIENCES
Rue de Bruxelles 61 - 5000 NAMUR
Tél. 081/72.41.11 - Telex 59222 facnam-b - Telefax 081/23.03.91

Mesure de densité en analyse des données : algorithmes, programmation C et comparaison de méthodes

Emmanuel BOURGEOIS

Résumé

Le problème de classement a, en analyse des données, retenu notre attention : sous l'hypothèse d'une réalisation d'un processus de Poisson non homogène, l'individu à classer est affecté à la "référence" la plus "dense" autour de lui.

Le sens précis apporté à la notion de "densité" définit un besoin particulier : il est nécessaire de développer un algorithme efficace pour le calcul du nombre de points, parmis une base d'un espace de dimensions quelconque, tombant dans un hypercube de mesure h centré en le point à classer.

Nous discutons les algorithmes associés à principalement deux méthodes. Le programme CES, en langage C, autorise une comparaison à l'issue de laquelle nous ouvrons la perspective d'une solution "optimale" à la frontière des deux méthodes discutées.

Mémoire de licence en Sciences Mathématiques
Année académique : 1991 - 1992
Promoteur : Professeur J.-P. RASSON

TABLE DES MATIERES

Chapitre 3 : Mesure de la densité ...

...d'une classe autour d'un individu

Le champ de la discussion
Le problème
La méthode naïve
La méthode projective
- Principe
- Illustration
- Développement
  - Première étape : repérage des points "suspects"
    - Formalisation
    - Algorithme : tr_ADD, tr_SIZ et ORGA
  - Seconde étape : Test sur les points "suspects"
    - Formalisation
    - Algorithme
  - Intégration des deux étapes

La méthode naturelle
- Principe
- Illustration : "dim" étapes actives
- Algorithme
  - La structure str_aux
  - Opérations sur str_aux
    - Tri rapide
    - Recherche dichotomique
  - Utilisation pour l'accomplissement d'une étape
    - Qsort
    - Ai-excess
    - Ai-default
  - Intégration des "dim" étapes

Clôture

Le programme CES
Comparaison des méthodes
Conclusions et perspectives

Avant propos

1 Statistique, analyse des données et informatique

La statistique est un domaine si vaste qu'il est impossible d'en donner une définition générale. Les méthodes en sont aujourd'hui utilisées dans presque tous les secteurs de l'activité humaine et font partie des connaissances de base de l'ingénieur, du gestionnaire, de l'économiste ... Le travail du statisticien peut ainsi prendre des formes très variées. Il est cependant commode de les regrouper en trois grandes catégories. Le recueil des données fait appel à diverses techniques différemment productives que nous n'aborderons pas. Après le recueil la démarche statistique consiste d'une part à traiter et d'autre part à interpréter les informations.

Les trois catégories de travaux statistiques que l'on vient de mettre en évidence sont loin d'être exclusives l'une de l'autre. Au cours de l'histoire, la collecte d'observations et la méthodologie de leur emploi se sont développées de façons largement indépendantes. Il serait vain, pourtant, de recueillir des données, si ce n'était pour les traiter et les interpréter en vue d'éclairer les actions humaines ou de faire progresser la connaissance des phénomènes. Inversément, la manière de recueillir des données peut et doit être influencée d'abord par les méthodes de traitement ultérieures et ensuite par l'utilisation pratique de ces données ou des produits qui en sont dérivés. Ces problèmes de frontières sont très complexes et il n'est pas ou guère possible de les discuter en détail dans un bref avant-propos; en revanche il est possible et sans doute utile de préciser un peu le double aspect de cette démarche statistique : l'aspect descriptif ou exploratoire et l'aspect inférentiel ou décisionnel.

La notion fondamentale en statistique est celle d'ensemble d'objets équivalents que l'on appelle population. Ce terme hérité des premières applications de la statistique qui concernaient la démographie est employé pour désigner toute collection d'objets à étudier ayant des propriétés communes. Ces objets sont appelés des individus. Chaque individu d'une population est décrit par un ensemble de caractéristiques, appelées variables, qui peuvent être selon leur nature d'une part quantitatives, discrètes ou continues, sur lesquelles les opérations arithmétiques courantes ont un sens et d'autre part qualitatives lorsqu'elles s'expriment par l'appartenance à une catégorie. La statistique traite plus des propriétés des populations que de celles d'individus particuliers. Généralement la population à étudier est trop vaste pour pouvoir être observée exhaustivement : c'est évidemment le cas lorsque la population est infinie. Lorsque l'on n'observe qu'une partie de la population la partie

étudiée s'appelle échantillon. La statistique inférentielle a pour but d'étendre les propriétés constatées sur l'échantillon à la population toute entière et de valider ou d'infirmer des hypothèses a priori ou formulées après une phase exploratoire. Le calcul des probabilités est un outil essentiel à cette extrapolation mais on ne peut y réduire la statistique parce que celle-ci, à côté du calcul des probabilités, utilise essentiellement des outils mathématiques assez classiques et de plus en plus l'informatique.

Le développement des moyens automatiques de production d'information a progressivement noyé le statisticien sous une masse croissante de résultats, une "quantité d'information" trop volumineuse pour l'esprit humain. A supposer qu'il soit possible de les retenir par coeur, ce n'est pas cela qui peut procurer une connaissance opératoire. La statistique exploratoire a pour but de synthétiser l'information contenue dans les données en extrayant de leur fatras les quelques renseignements vraiment intéressants qui construiront cette connaissance. Certes on pourrait les traiter en usant des procédés habituels de la statistique descriptive : description (unidimensionnelle) séparée des résultats obtenus pour chaque variable. Ces procédés d'une grande utilité pratique et, à première vue du moins, d'une grande simplicité intellectuelle, permettent d'accéder, progressivement, à une meilleure visualisation des données qui facilite leur interprétation. Dans la plupart des applications cependant on observe non pas une variable par individu, mais un nombre p souvent élevé. L'étude séparée de chacune de ces variables est une phase indispensable dans le processus de dépouillement des données mais tout à fait insuffisante : en effet l'étude séparée de chaque variable laisse de côté les liaisons qui peuvent exister entre elles et qui sont souvent l'aspect le plus important. Un développement de la statistique descriptive était nécessaire pour étudier les données en tenant compte de leur caractère multidimensionnel. L'analyse des données, brûlant les étapes de la statistique descriptive classique, permit d'obtenir ces visualisations d'un type nouveau.

Discipline jeune - puisqu'elle n'est encore que dans la phase du développement et des découvertes - l'analyse des données est actuellement chez les statisticiens, l'objet d'un véritable phénomène de mode, caractérisé pourtant à la fois par l'engouement et le rejet. Sa vocation principale n'est pas d'être un objet de méditation et de construction théorique pour le mathématicien, mais d'être un instrument fait pour la main du praticien et utilisé quotidiennement dans la pratique; elle se situe en aval de la production des résultats et immédiatement en amont de leur présentation littéraire qu'elle prépare, des raisonnements probabilistes que nous pouvons effectuer sur eux et des études proprement économiques,

sociologiques ou autres qu'ils peuvent nourrir. Méthode descriptive elle n'explique rien par elle-même; mais elle conduit le théoricien à se poser les questions opportunes.
Servant à lutter contre le gachis de collecte, cette plaie de la statistique qui consiste à gaspiller une information rassemblée à grands frais mais que l'on est incapable de déchiffrer et d'interpréter, elle vise non pas à produire des significations mais seulement, dans une zone intermédiaire à mi-chemin entre le concret et l'abstrait, à faciliter leur dégagement, en présentant le corpus des données sous une forme appropriée. Certes l'analyse des données restitue des évidences. Comment pourrait-elle ne pas les restituer si ces évidences sont exactes, c'est-à-dire correspondent à des faits réels et importants ? Cependant, pour celui qui sait s'en servir, elle révèle en outre rapidement des phénomènes fins qu'il aurait été difficile de déceler par les méthodes classiques sous un long travail.

L'apparition des moyens de calcul automatique et les développements récents de l'informatique ont révolutionné la pratique de la statistique en rendant possible un traitement descriptif plus rapide et plus sûr, sans trop les appauvrir, de grands gisements de données. Il n'existerait, en effet, pas d'analyse des données sans informatique - en raison du volume des calculs, nécessaires à la prise en compte de données multidimensionnelles, que seul l'ordinateur peut réaliser -. La mise au point d'un procédé d'analyse des données réclame donc un double effort d'initiation aux méthodes existantes d'une part, de domination des outils informatiques d'autre part et en particulier l'apprentissage d'un langage de programmation. Pour assouplir l'usage de l'analyse des données, pour l'insérer dans le rythme du travail quotidien des techniciens et aussi pour donner à ses produits une allure attrayante, d'importants progrès sont encore nécessaires, auxquels nous ambitionnons d'apporter, au travers de ce mémoire, notre participation.

* — *

2 Structure du document

L'origine de ce travail a été une prise de conscience : prise de conscience de ce que si le langage Fortran constituait à sa sortie un formidable outil pour le programmeur, tant de choses se sont, depuis, passées dans le domaine informatique que le scientifique qui continue aujourd'hui d'utiliser Fortran se prive des importants progrès réalisés pendant plus de trente années d'évolution. Il nous a, de ce fait, paru impératif de montrer combien il était opportun pour le programmeur scientifique avide de performances, en particulier pour le statisticien travaillant en analyse de données, d'utiliser un langage de type "moderne". Ces propos sont généreusement développés dans le chapitre premier où nous présentons, en outre, le langage que nous avons choisi, à savoir le langage C.

Parmis les procédés d'analyse de données, l'analyse discriminante, en particulier le classement, parmis plusieurs classes de référence, d'un individu supplémentaire, a retenu notre attention. Dans le chapitre deux nous montrons comment l'exploitation d'un nouveau modèle, basé sur l'hypothèse d'une réalisation d'un processus de Poisson non-homogène dans l'union de K domaines convexes, permet le classement d'individus que la règle issue du modèle existant considérait, pourtant, comme inclassables : il s'agit des individus associés aux points de l'espace appartenant aux intersections de ces domaines. La règle développée affecte l'individu supplémentaire à la classe la plus "dense" autour de celui-ci. Le sens précis apporté par la méthode à la notion de "densité" définit un besoin particulier : il est nécessaire de concevoir un outil algorithmique efficace pour le calcul $\forall\, i \in \{1, \cdots, K\}$ du nombre n_i de points, parmis N_i de $i^{\text{ème}}$ classe de référence, "tombant" dans un hypercube H de mesure h centré en le point à classer.

Le chapitre trois présente et compare les algorithmes associés à principalement deux méthodes pour résoudre le problème issu de ce besoin précis.
L'une "projective" est particulièrement bien adaptée à l'utilisation des seuls concepts fournis par une programmation en un langage du niveau technique de Fortran.
Le principe en est achevé en deux étapes : dans un premier temps, une projection, sur le plan principal des deux premières coordonnées, de l'espace total opère une sélection qui écarte des points dont on est certain qu'ils ne peuvent tomber dans l'hypercube : il s'agit des points dont les projections "tombent" à côté du pavé $\overline{H}$, image par la même projection de H. Un test exact définitif est dans un second temps effectué sur chacun des m_i points restants ($N_i \geq m_i \geq n_i$) qui détermine leur (non-)convenance effective.

L'algorithme compense fortement la pauvreté des moyens par la mobilisation d'une importante mémoire. L'implémentation sur le puissant système VAX en a malgré tout révélé un inconvénient majeur, théoriquement prévisible : le temps nécessaire à l'exécution croît de manière quadratique avec la mesure h de l'hypercube de sorte que si les temps d'exécution sont réjouissants avec h petit, dès que h grandit ils deviennent vite gênants.
Le second algorithme utilise pleinement la souplesse et la variété des moyens techniques mis à disposition du programmeur par un langage moderne tel que C.
La méthode divise l'action en d étapes où d est la dimension de l'espace total. La $j^{\text{ième}}$ étape ($1 \leq j \leq d$) rejette les points dont la $j^{\text{ième}}$ coordonnée est telle que ce point ne peut appartenir à H. Les points forts de chaque étape sont au nombre de trois : un tri rapide effectué sur les $j^{\text{ième}}$ coordonnées et deux séries d'appels successifs à une procédure de recherche binaire. L'une (des deux séries) aboutit à la définition d'une limite supérieure, l'autre à une limite inférieure acceptable pour les $j^{\text{ième}}$ coordonnées : c'est à dire que les points dont la $j^{\text{ième}}$ coordonnées est comprise entre ces deux limites sont considérés à l'étape $(j+1)$. Le nombre de points considérés diminue d'étape en étape. Les points qui restent au delà de la $d^{\text{ième}}$ et dernière étape sont les points qui tombent dans l'hypercube.

Le programme CES réalisé avec turbo C 2.0. intègre les deux procédés au sein d'un même environnement. De la comparaison issue de CES il ressort que l'utilisation, dans le cas de h "grand" de la méthode naturelle (algorithme 2) permet d'éviter la croissance abusive du temps d'exécution. Il y a malheureusement une ombre au tableau de l'efficacité de cette alternative à la méthode projective (algorithme 1) : le prix à payer est une limitation du nombre de points à classer. Ces résultats constituent l'aboutissement (la clôture) de ce mémoire. En guise de conclusion nous ouvrons la perspective d'une méthode "meilleure" quelque part à la frontière des principes sur lesquels se basent les méthodes traitées dans ce travail. Nous ne pouvons qu'encourager dans cette voie de recherche toute personne prête à relever le défi de l'efficacité optimale.

* — *

3 Remerciements

Au terme de cette présentation, je tiens à exprimer toute ma reconnaissance à Monsieur le Professeur Jean-Paul RASSON, mon promoteur. Son appui, ses encouragements et son aide patiente m'ont permis de réaliser ce travail.

Que Monsieur Vincent GRANVILLE soit également remercié de sa participation à mes travaux : sa disponibilité, ses conseils pratiques et son expérience de la programmation et du langage C.

Enfin, je n'oublie pas que mes parents et amis m'ont aidé et soutenu, chacun à leur manière, durant ces années; je les en remercie chaleureusement.

...à Namur, en août 1992 - E. BOURGEOIS.

Chapitre 1 :
Le chapitre informatique

1 Avertissement

L'informatique, depuis ses balbutiements, a considérablement évolué. Cette évolution s'est orientée dans deux directions; d'une part une augmentation des possibilités technologiques de l'ordinateur, d'autre part une souplesse et une facilité plus grande dans son utilisation. Cette seconde voie vise à permettre à tout un chacun d'utiliser un ordinateur. Elle a été réalisée en définissant des langages dits de programmation.

L'expérience ayant montré que l'on pouvait créer divers langages de programmation relativement adaptés à l'expression de méthodes propres, les langages se multiplièrent avec les méthodes. Dans les milliers de langages qui virent le jour, quelques-uns émergent qui provoquent des réactions passionnelles. Il se développe autour de chacun une secte d'adorateurs prêts, à tout moment, à mettre en évidence un problème qui s'exprime mal dans les autres langages tandis que dans "le" langage "un tel" tout va bien. Mais pendant que l'on se bat ainsi pour les langages on oublie l'essentiel.

Les langages sont faits pour dire quelque chose. La façon de dire importe mais quand on s'intéresse à ce qui sort de l'ordinateur, ce n'est pas elle qui compte, c'est ce que l'on a dit, l'organisation du discours et la forme de la pensée.

Si nous faisons notre ces propos c'est parce que nous voulons empêcher le lecteur qui parcours ce chapitre de considérer les pages suivantes comme l'expression manifeste de l'une de ces querelles de chapelle entre utilisateurs de différents langages. Que nos intentions soient correctement ressenties : il ne s'agit pas d'arbitrer, de manière extrêmement partiale, un match Fortran contre C, ou C contre Fortran, puisque c'est de ces deux langages dont il est question. Ce dont il s'agit c'est de ne pas nier les progrès réalisés en matière de programmation depuis la création de Fortran (et ceci se passait dans des temps très anciens ...), de comprendre que ce géant du passé n'est plus utilisé aujourd'hui que parce qu'il était là avant les autres et de renverser les barrières que cette utilisation excessive a dressé et continue de dresser sur notre chemin.

* — *

2 Le langage FORTRAN

Fortran, né en 1954, est le plus vieux, et peut-être le plus puissant des langages de haut niveau survivants. Au travers des années, la famille Fortran a été le langage le plus largement utilisé dans les applications scientifiques et d'ingénierie. Le style du langage Fortran a, de plus, engendré bon nombre de dialectes, chacun correspondant à un besoin différent de "plier" le langage à quelques cas particuliers.

2.1 Bref historique de FORTRAN

Le nom "Fortran" est une abréviation de "FOR-mula TRANslation" (traduction de formules) : Fortran fut un des premiers langages permettant de coder un calcul arithmétique en une seule instruction, par une formule ressemblant à celles que l'on peut écrire en mathématiques. Le codage de formules était d'ailleurs le but initial du projet Fortran; après quelques temps seulement, ses membres s'aperçurent qu'il suffisait de peu de choses pour spécifier dans le même système l'agencement du calcul, donnant ainsi naissance au premier langage de programmation de grande diffusion.

Le premier membre de la famille, Fortran I, est né en 1954 et fut implémenté sur IBM 704 en 1956. Deux ans plus tard Fortran II voyait le jour. Il comprenait un nombre significatif de perfectionnements par rapport à Fortran I, incluant la définition des sous-programmes et leurs appels. Entre 1958 et 1963, Fortran fut implémenté sur plusieurs calculateurs. Fortran III fut développé pendant cette période, mais il comportait trop de caractéristiques dépendant de la machine pour être d'un usage public.

En 1962, Fortran IV fut développé pour IBM 7090/7094. Pendant la même année, l'American Standards Association (ASA) forma un comité pour définir une version standard de Fortran. Cet effort était la réponse aux différentes versions qui commençaient à proliférer parmis les diverses implémentations du langage, ce qui rendait impossible tout transfert d'un calculateur à un autre, avec la certitude d'un déroulement correct du programme. Ce comité fournit deux versions standards de Fortran en 1966. L'une, appelée tout simplement "Fortran", était proche de Fortran IV, l'autre fut appelée "Basic Fortran".

En 1977, des extensions importantes sont ajoutées à Fortran et le standard est modifié en conséquence. Cette version, connue comme Fortran 77, ajoute des fonctions améliorant son usage en programmation structurée, dans le traitement des données et des fichiers. Comme pour le standard 1966, il existe aussi un sous-ensemble de Fortran 77. De plus Fortran 77, proprement dit, contient la version 66, de sorte que la compatibilité ascendante est conservée pour des programmes qui ont été écrits selon l'ancien standard.

2.2 Implémentations et variantes de FORTRAN

Fortran 77 existe aujourd'hui sur presque tous les calculateurs communément utilisés. Les implémentations de Fortran sont conformes à la version standard. Les plus importantes divergences résultent de la taille des mots-machine et de la représentation des nombres, qui concerne la précision des calculs mathématiques. D'autres différences apparaissent quand diverses implémentations ajoutent des extensions au langage, limitant ainsi la portabilité des programmes. En effet, si Fortran a atteint un degré de normalisation supérieur à celui d'autres langages de grande diffusion, malheureusement la norme officielle du langage présente des lacunes importantes et presque tous les constructeurs proposent une version de Fortran "améliorée" par de nombreuses extensions, dont l'effet le plus clair est de rendre hasardeux tout transfert d'un programme d'une machine à une autre.

2.3 Principales applications de FORTRAN

Presque tous les usages de Fortran sont du domaine scientifique et technique. Certaines extensions permettent l'emploi de FORTRAN dans la gestion des données et le traitement de texte, mais il n'est pas considéré comme un remplaçant sérieux des outils spécialement conçus pour ces domaines d'applications.

* — *

3 De l'opportunité d'un passage de FORTRAN à C

Pas 1 : FORTRAN est techniquement dépassé.

L'une des caractéristiques principales d'un processus de fabrication industrielle est la possibilité d'effectuer un véritable contrôle de la qualité. Il doit donc exister, dans toute discipline d'ingénierie digne de ce nom, une définition simple de la qualité, communément admise par les gens du métier, et des méthodes bien établies pour garantir cette qualité.

Que la construction de programmes s'apparente à une discipline d'ingénierie n'est pas encore chose universellement admise, encore que le succès croissant de l'expression "génie logiciel" témoigne de l'attente à laquelle répondent les efforts menés en ce sens. Un aspect important de cette évolution est l'ensemble des réflexions entreprises depuis quelques années pour aboutir à une bonne définition de la notion de qualité en logiciel.

Une vue large de cette qualité embrasse tout aussi bien les spécifications, les dossiers d'analyse et de conception, la documentation remise aux utilisateurs, les délais de réalisation, le coût du projet, l'efficacité du système, sa facilité d'emploi, $\cdots$ et non seulement les caractéristiques du texte des programmes.

Il est, à ce titre, vivement recommandé, en programmation Fortran, de s'en tenir à la norme du langage, sous peine de perdre l'avantage essentiel de l'utilisation du Fortran. La pratique montre cependant que ce conseil est difficile à suivre (...qu'il est peut-être impossible d'écrire un programme complètement portable ...) et que son application stricte, qui rend le travail des programmeurs tellement pénible et les programmes si peu efficaces, n'est pas toujours un souci naturel chez des gens pour qui la programmation n'est pas en général une fin en soi mais plutôt une étape obligée dans la résolution de problèmes très divers.

Les auteurs des procédures d'analyse des données réalisées depuis plusieurs années au sein du département de mathématique (unité de statistique) des Facultés Universitaires de Namur, se sont ainsi préoccupés, de la qualité du style de leurs programmes et de l'efficacité de ceux-ci mais peu de la portabilité en laquelle une perte a été engendrée par l'emploi d'extensions à la norme du langage.

Le langage de programmation (Fortran) en lequel ont été écrits ces programmes mérite en outre, encore quelques commentaires. Fortran, défini en 1954, est le plus ancien des langages de programmation utilisés aujourd'hui. De tous les points de vue, ce langage est techniquement dépassé; tant de choses se sont passées en informatique depuis trente ans que la survivance d'un tel témoin constitue l'un des paradoxes de l'histoires des techniques que la force de l'habitude peut seule expliquer, et dont un autre exemple est le maintien jusque sur les terminaux d'ordinateurs des claviers Azerty ou Qwerty, où l'arrangement des lettres, conçu à la fin du XIXè siècle pour éviter les incidents mécaniques sur les premières machines à écrire, tend explicitement à maximiser la distance entre les symboles les plus couramment utilisés ensemble.

Entendons-nous bien : Fortran en 1954, constituait une réalisation d'une portée considérable, surtout si l'on se souvient que le premier compilateur fut diffusé en même temps que le langage, et qu'il produisait un code objet de grande qualité. La rugueuse simplicité (qui à sa naissance fit la force de Fortran) du langage a en outre malheureusement été remise en cause par son évolution ultérieure (résultat des travaux de l'ASA aujourd'hui appelée ANSI, organisation américaine de normalisation) : moins par la norme publiée en 1966 que par celle de 1978, dite Fortran 77.

On peut cependant encore reconnaître à Fortran, dans son état actuel , certaines qualités : un reste de la frugalité ancestrale; l'accent mis sur la compilation séparée, utile au développement de sous-programmes; la faculté de créer de véritables "unités de données" grâce à la notion de commun, et la possibilité d'écrire des programmes portables si l'on respecte strictement un grand nombre de règles. Mais c'est à peu près tout. En 1954, l'étude de la programmation, des algorithmes, des structures de données, des langages ne faisait que balbutier; des milliers de chercheurs et de praticiens ont, depuis cette époque, amassé autour de ces sujets une quantité considérable de connaissances. L'ingénieur qui programme aujourd'hui en Fortran se prive de toute cette évolution. C'est un peu comme s'il écoutait Radio 21 sur un poste à galène. Rien de ce qui existe dans les meilleurs langages d'aujourd'hui, mécanismes de description de structures de données complexes, vérification statique des types, création dynamique d'objets, récursivité, outils de programmation modulaire, généricité, ... ne se rencontre en Fortran.

Fortran reste, pourtant, l'un des principaux langages et, tout particulièrement, régit en maître à peu près incontesté le domaine du calcul scientifique. Cette situation est, nous l'avons dit, moins due à une passion générale pour ce langage qu'au poids des habitudes et du logiciel existant. En fait, le maintien de ce langage semble dû essentiellement aux problèmes économiques qu'entraîneraient la conversion des importantes "bibliothèques de programmes" existantes, la réécriture des compilateurs "optimisateurs" auxquels des efforts considérables ont été consacrés, ... de sorte que même si quelques-uns des esprits les plus éclairés de la communauté des Fortranateurs se posent aujourd'hui quelques questions, qu'on le déplore ou non, des programmes Fortran continueront donc d'être écrits pour un certain temps encore.

Un certain temps ... suffisant sans doute pour ne pas abandonner encore le langage Fortran dans les travaux qui se poursuivent mais aussi un certain temps ... trop court sans doute que pour ne pas préparer dès à présent cette révolution des habitudes, si attendue, justifiée et crainte à la fois qu'elle ne peut ne pas se produire ... un temps prochain. C'est ce que nous avons compris. Nous avons donc décidé de développer en analyse des données, en parallèle de la voie traditionnelle de programmation Fortran des outils informatiques, une voie de programmation moderne de ces mêmes outils en un langage récent : le langage C. Le temps venu de l'éveil des programmateurs scientifiques, les jalons posés aujourd'hui et les efforts consentis nous placerons sans aucun doute en position avantageuse.

* — *

4 Le langage C

Le langage C est conçu pour encourager une économie d'expression dans une grande variété d'applications. Le premier usage de C concerne les systèmes d'exploitation; ainsi, 90 % des codes d'UNIX ont été générés à partir de C. Toutefois, les opérateurs C, les types structurés de données et la bibliothèque de fonctions furent, dès les premières utilisations du langage, si exceptionnellement complets que son domaine d'application ne put que s'étendre.

Ce langage n'est ni d'un "très haut niveau", ni très riche; il ne se spécialise pas non plus dans un champ particulier d'applications. Mais l'absence de restrictions et son côté général le rendent plus adapté et plus efficace dans beaucoup de cas que des langages supposés plus puissants.

4.1 Bref historique de C

Il est difficile de dissocier un langage tel que C du système UNIX qui lui servit tout d'abord de banc d'essai, puis de référence, pour en constituer ensuite l'un des principaux vecteurs de diffusion.

En 1969, les laboratoires Bell cherchèrent un remplaçant au système d'exploitation Multics des calculateurs PDP-7. Une première version d'UNIX fut ainsi écrite en langage assembleur. Durant la même période, un langage expérimental était développé par Kenneth Thompson, inventeur d'UNIX. La majeure partie des principes fondamentaux de ce langage, appelé B, est issue du langage BCPL développé par Martin Richards. En 1972, le langage C fut conçu par B.W. Kernighan et D.M. Ritchie comme une extension de B, la principale différence étant que C disposait d'une importante collection de types standards dont les langages BCPL et B étaient essentiellement dépourvus.
Peu de temps après, en 1973, UNIX lui même fut substantiellement étendu, et plus de 90 % de la nouvelle version écrits en C. Le pari réussi d'UNIX fut de rendre disponible sur un mini-ordinateur des fonctions qui n'étaient alors remplies que par de très gros systèmes d'exploitation. En raison de son indépendance à l'égard du langage assembleur, UNIX (et bien sûr C), grâce au développement par S. Johnson d'un compilateur aisément "portable", fut rapidement installé sur une grande variété de calculateurs, et acquit bientôt un domaine d'applications étendu. Le pari initial était ainsi couronné d'un succès jamais démenti.

En 1976-1977, UNIX fut adapté au système VAX et, indépendamment, une nouvelle version fut développée à l'université de Californie à Berkeley. La fin des années 70 et les années 80 ont été riches en adaptations, sur diverses machines de toutes puissances, d'UNIX et de C dans un premier temps, ensuite principalement de C lorsque celui-ci devient indépendant du contexte d'UNIX. Baptisé "langage de programmation des ordinateurs" parce que employé pour écrire des systèmes d'exploitation, du fait de son niveau proche de la machine, C fut ainsi également utilisé lors de l'écriture de programmes en majorité numériques, de traitement de textes et de base de données.
Les compilateurs sont disponibles pour beaucoup de machines sous leur propre système d'exploitation. La montée en puissance des micro-ordinateurs, tels que l'IBM-PC et les "compatibles", a permis son implantation sur ce type de matériel, sous DOS.

4.2 Implémentations et variantes de C

Depuis la première utilisation sur PDP-7, on a retrouvé C sur divers systèmes. Il existe aussi plusieurs marques de compilateurs C. Mais dès sa sortie le TURBO C de Borland a bénéficié d'une véritable ruée justifiée en partie par son prix mais surtout par des qualités hors du commun. Tandis que les compilateurs C traditionnels progressent laborieusement, les opérations de compilation et d'édition de liens avec TURBO C sont simplifiées grâce au pilotage par menus et exécutées automatiquement, de la même façon qu'avec les autres turbo-compilateurs connus. Les délais de compilation sont spectaculaires et le code généré, largement optimisé, des plus véloces qui soient. Turbo C correspond à la norme de l'ANSI (American National Standards Institute).

La version 2.00 de Turbo C constitue un "must" dans le monde "Turbo", par l'extraordinaire ajout d'un débogueur symbolique d'une rare qualité laissant toute latitude pour tester, expérimenter et perfectionner les programmes. Peu de compilateurs offrent un outil de développement aussi sophistiqué, et, en outre, la manière de l'utiliser est à notre avis parmis ce qui se fait de mieux, de sorte que nous sommes heureux d'avoir choisi le Turbo C 2.00 comme support.

Le texte de référence pour C est l'oeuvre de Kernighan et Ritchie qui donnent la description la plus connue du langage C; Turbo C est conforme à la définition de Kernighan et Ritchie (cfr. "Le langage C", manuels informatiques Masson).

4.3 Principales applications de C

Bien qu'il soit (à l'origine) un langage de programmation de systèmes, C, ne se spécialisant dans aucun champ particulier, a acquis une grande popularité dans différentes applications. Ces dernières peuvent couvrir une étendue importante des différents domaines possibles, qu'il s'agisse de la gestion, des programmes scientifiques, de la C.A.O., des jeux et bien sûr de la programmation système.

* — *

5 De l'opportunité d'un passage de FORTRAN à C

Pas 2 : C est un langage de type moderne.

Le langage C n'est pas plus difficile qu'un autre langage évolué. Sa réputation de difficulté n'est qu'apparente et tout un chacun peut le pratiquer, même s'il n'est pas ingénieur informatique. En fait, sa complexité apparente est surtout due à la richesse de ses opérateurs et au fait que les habituelles barrières de sécurité ont été volontairement omises, afin de supprimer d'éventuelles contraintes d'utilisation. Il laisse une liberté pratiquement totale au programmeur, qui ne rencontre pratiquement aucun frein. En C il existe toujours une solution à une situation épineuse. Sa souplesse est sans rivale, en ce qui concerne la manipulation des données. C'est ce qui justifie son emploi pour la programmation d'interfaces de toute nature, ainsi que de systèmes d'exploitation, entre autre. Cette (quasi) absence de contraintes peut engendrer des erreurs difficiles à retrouver, surtout lorsqu'on aborde ce langage pour la première fois car des effets indésirables peuvent être induits et non détectés par le compilateur. Son emploi nécessite donc de la prudence, une bonne compréhension du langage, une approche progressive de son apprentissage et surtout la mise en pratique de l'enseignement reçu. Des bases saines sont absolument nécessaires à la domination de ce langage performant. Mais avec l'habitude de sa pratique et une programmation rigoureuse, plus qu'avec les autres langages, il demeure quand même simple à utiliser.

Les propos suivants se proposent d'aider le lecteur à cerner les principes essentiels du langage C sans constituer une initiation. Nous supposons connus, assimilés, un certain nombre de concepts de programmation et essaierons seulement d'être complet sans être précis afin de ne pas être submergé de détails, de lois formelles ou d'exceptions.
Voulant atteindre, le plus vite possible, le moment où le langage C se révèle plus adapté dans beaucoup de cas que des langages supposés plus puissants, en particulier Fortran, nous avons cherché à éviter le piège d'une exhaustivité écrasante et n'avons, pour ce faire, développé que les aspects du langage jugés les plus significatifs de son efficacité.

Le langage C est un langage de type moderne, aussi bien en ce qui concerne la structuration des programmes que la représentation des données.

Les éléments principaux utilisés par tout programme sont les constantes et les variables. Afin de pouvoir manipuler aisément le contenu des variables, le langage C, comme beaucoup, a prévu de les identifier par un nom. Ce dernier est donc appelé identificateur. C base l' interprétation d'un identificateur sur deux caractéristiques : sa "classe de mémorisation" et son "type". Les classes de mémorisation, déterminant l'endroit et la durée de vie de la mémoire associée à un identificateur, autorisent la déclaration de variables globalement ou localement. Le type permet de définir les valeurs se trouvant en mémoire.
Le langage C offre une relative simplicité au niveau de la définition de ses types de données de base puisque les objets que le langage manipule sont de types très peu nombreux. Ils se limitent, en effet, essentiellement à des caractères, trois représentations numériques et à leurs adresses respectives. Heureusement cela ne veut pas dire que le programmeur ne dispose que de ceux-là, pour l'ensemble de ses applications. Les variables doivent être déclarées avant d'être utilisées bien que certaines déclarations soient faites implicitement par le contexte. Ces déclarations dressent la liste, en indiquant leurs types et parfois leurs valeurs initiales, des objets nécessaires au développement de la programmation. Selon la nature de cette dernière, il sera possible d'apporter certaines modifications aux types de base, de façon à simplifier la résolution de tout problème ayant trait à la gestion, au calcul scientifique ou au graphisme. C'est ainsi qu'il est possible de créer de nouveaux types de données tels que des tableaux, des chaînes de caractères, des ensembles, des listes, ..., soit en apportant ses propres définitions, même si elles s'appuient sur ce qui existe déjà, soit en regroupant subtilement des entités déjà connues à l'aide des structures et/ou des unions principalement mais aussi grâce aux notions de "fonction" et de "pointeur", qui plus que tout autre font la réputation de C. En général, on peut appliquer ces méthodes de construction d'objets, de façon récursive.

L'instruction "typedef" permet l'introduction de nouveaux concepts, avec les données. Elle offre la possibilité d'employer, avec certitude, des variables en concordance absolue de type pour représenter des objets de même "nature". Une autre possibilité intéressante consiste à recréer le type booléen, qui fait défaut en C; en C l'opération de transfert du contenu d'une variable à l'intérieur d'une autre variable est légale même s'il s'agit de deux variables de type différent de sorte qu'il est préférable de considérer le type caractère comme pouvant mémoriser n'importe quelle valeur numérique : une conséquence directe de cette constatation est qu'il est très facile de simuler une variable booléenne en utilisant une simple variable de type caractère à laquelle on attribue systématiquement une valeur numérique égale à 0 ou 1. Par convention 0 sera destiné à constater un état FAUX et 1

un état VRAI. Comme une condition vérifiée, en C, correspond à une valeur numérique différente de 0, cela permet de simplifier aisément l'écriture des conditions de tests.
Une structure est une collection de données, regroupée logiquement au sein d'une nouvelle entité définie par le mot-clé STRUCT. Il n'est pas possible d'effectuer des opérations d'une façon globale sur une structure, mais par contre chacun de ses membres est considéré comme une variable classique accessible.
Une union permet de définir une zone de mémoire unique, pouvant recueillir des données de types différents, selon les besoins. L'emploi d'une telle définition permet d'économiser de la place en mémoire; il faut cependant reconnaître que, parce que rien ne permet d'affirmer avec certitude le type de données qui peut s'y trouver à un moment précis, c'est au détriment de la sécurité de programmation.
La combinaison des deux définitions de structure et d'union peut donner un résultat intéressant. L'utilisation en est judicieuse, lorsqu'elle a pour but de découper une même zone mémoire en plusieurs sous-zones et d'avoir accès à ces dernières d'une façon directe.

Le langage C est un langage dont le "noyau", qui ne comporte qu'un nombre réduit d'instructions, est demeuré pratiquement tel que son concepteur l'a défini. Avec C l'essentiel des tâches s'effectue par des appels à des fonctions préprogrammées, développées par le programmeur ou encore ajoutées par lui à la bibliothèque déjà existante. Les fonctions utilisent des variations à l'infini du noyau d'instructions pour en créer de nouvelles : regroupement d'actions permettant la sortie d'un résultat. Ces dernières serviront à leur tour à composer de nouvelles instructions et ainsi de suite. Ainsi la puissance du langage devient rapidement phénoménale tout en laissant les sources des programmes très portables. Cela signifie que théoriquement ils sont capables de tourner sur n'importe quelle machine sans modification sensible. En réalité, certaines fonctions qui composent l'environnement de C, extrêmement dépendantes de la machine, seront à réécrire. Mais le "coeur" du langage reste universel et les fonctions seront toujours employées de la même façon par le programme puisqu'il est possible de s'en servir comme d'une boîte noire, sans connaître sa composition interne en s'occupant seulement de son point d'entrée et de son point de sortie; même si le contenu doit s'adapter, si nécessaire, à la machine concernée de sorte que la dite fonction effectue correctement le travail qui peut lui être demandé.
Le langage C fournit les structures fondamentales nécessaires à l'élaboration de programmes structurés; les agencements des déclarations, les instructions itératives ou "boucles" et les instructions conditionnelles en lesquelles C est particulièrement riche, comme il est riche en opérateurs associés à ces instructions. Mais il y a une différence fondamentale entre le

langage C et les autres langages évolués : il est d'usage de dire que C est un langage de bas de gamme. Ce n'est nullement péjoratif, car cette définition indique simplement qu'il est plus proche de la machine, du langage machine. Cela signifie que comme dans le cas de l'assembleur, les contrôles effectués par le compilateur sont peu nombreux, mais qu'en contrepartie il est imbattable au niveau de la vitesse d'exécution des programmes.

Les langages évolués permettent, pour la plupart, le découpage d'un programme en sous-parties, accomplissant chacune une action complètement définie à son niveau. En Fortran il s'agit de "subroutines" et en Pascal de "procédures". Le langage C ne fait pas exception à la règle et permet également de concevoir des programmes modulaires, facilitant la lecture et la compréhension des différentes actions qui aboutissent à la mise en oeuvre d'une action complexe. Suivant les règles de la programmation structurée, chaque niveau du traitement peut-être décomposé en sous-niveaux, supposés déjà développés. Puis chacun de ces sous-niveaux peut à son tour être décomposé en sous-sous niveaux et ainsi de suite, jusqu'au moment où tout a été décomposé en instructions C, du noyau ou issues de la bibliothèque. Lorsque l'on a atteint ce niveau le plus fin où les instructions élémentaires et indispensables ont été écrites, l'écriture du programme est terminée.
En C les sous-parties d'un programme ne sont composées que de "fonctions", sans faire appel à la notion de procédure. Cependant, si les procédures n'ont pas été prévues par les concepteurs du langage, il est possible de les simuler très facilement, en faisant en sorte qu'une fonction ne retourne aucun résultat, servant simplement, comme une procédure, à regrouper un certain nombre de commandes, répondant à un niveau de traitement identique, qu'il est commode d'individualiser. De sorte que, en fait, si le langage C ne reconnaît que des fonctions il y a cependant celles qui ne retournent aucun résultat et celles qui en fournissent un. Le langage C vise à rendre les fonctions efficaces et faciles à utiliser. Ainsi, un programme en langage C comprend, en général, un grand nombre de petites fonctions plutôt qu'un nombre restreint de fonctions de grandes tailles.
Les fonctions du langage C peuvent être utilisées de manière récursive, c'est-à-dire qu'une fonction peut s'appeler elle-même, soit directement, soit indirectement. Quand une fonction fait appel à elle-même, chaque appel crée un nouvel ensemble comprenant toutes les variables associées à cette fonction qui est alors indépendant de celui engendré précédemment. L'usage de la récursivité donne un programme compact, plus facile à comprendre.
Un programme peut se trouver dans un ou plusieurs fichiers sources; ceux-ci peuvent être compilés séparément et chargés en même temps, tout en utilisant des fonctions appartenant à des bibliothèques qui auront été compilées auparavant.

Une bonne part des erreurs faussement imputables au langage C peut être évitée une fois bien assimilés les concepts de "tableau", surtout en ce qui concerne les chaînes de caractères, et de "pointeur".

Un tableau est un ensemble d'éléments (variables, constantes ou ...) regroupées d'une façon séquentielle, dont le nombre est précisé mais les "positions" commencent à 0 et destinés à mémoriser des données, rigoureusement de même type, dont l'accès peut s'opérer de façon séquentielle et individuelle.
Un tableau peut comporter plus d'une dimension, sans limite à ce niveau, si ce n'est l'étendue de la mémoire, mais un tableau multi-dimensionnel est en réalité mémorisé et géré comme un tableau à une seule dimension, avec seulement un pseudodécoupage. Deux sortes de tableaux peuvent être rencontrés : les tableaux numériques et les tableaux de caractères, plus communément appelés chaînes de caractères. Le langage C représente en effet une chaîne de caractères comme un tableau de caractères. D'autres langages aussi, mais la différence est que C ne s'arrête pas, dans l'identification chaîne-tableau à la représentation. En C cette chaîne de caractères n'est plus une entité individuelle mais bel et bien un tableau, dont cependant la fin est indiquée par la présence d'un caractère nul, manipulable comme un tableau et, aurait-on tendance à dire, seulement comme un tableau, élément par élément. Evidemment, cette façon de procéder est laborieuse. C'est pour cette raison que la bibliothèque fournie renferme un certain nombre de fonctions prédestinées à travailler "globalement" sur des chaînes. Les principales manipulations portant sur des chaînes de caractères sont ainsi disponibles (lecture, écriture, copie, concaténation, extraction, ...).
Si les auteurs du langage C semblent avoir compliqué l'utilisation des chaînes de caractères à plaisir, il n'en est cependant rien. Bien au contraire, en C primo il n'y a plus aucune limite (sauf celle de la mémoire) à la longueur des chaînes de caractères (comme c'est le cas des autres langages de programmation), secundo il devient possible de les initialiser et tertio de faire pointer un pointeur sur une chaîne de caractères.

Le principe de la notion de pointeur repose essentiellement sur le fait que toute variable possède nécessairement une adresse en mémoire vive. Un pointeur est destiné à contenir uniquement l'adresse d'une variable, pas son contenu.
La notion devient particulièrement intéressante lorsque l'adresse d'un tableau est perçue comme étant constituée par l'adresse de la première variable le composant. En conséquence

l'adresse d'une chaîne de caractères sera en fait égale à l'adresse du premier caractère la constituant. Lorsqu'une telle adresse est connue, il devient simple de la référencier et, dans un second temps, d'accéder à la mémoire qui la suit.

En langage C, beaucoup de relations existent entre les pointeurs et les tableaux, des relations d'ailleurs assez fortes pour qu'il fut impératif de traiter simultanément les deux notions. En outre puisque les pointeurs sont des variables, on peut imaginer des tableaux dont les éléments sont des pointeurs. Toute opération utilisant un indiçage de tableaux peut ainsi être réalisée aussi grâce à des pointeurs. Le tableau de pointeurs occupe légèrement plus de place en mémoire et peut nécessiter une étape d'initialisation explicite, mais cette méthode présente deux avantages : premièrement, l'accès à un élément à l'aide d'un pointeur est achevé plus rapidement que grâce à un indiçage; secundo, les lignes du tableau peuvent avoir des longueurs différentes.
Précisons que ce concept de "pointeur" s'applique ainsi à toute collection complexe de variables et qu'il est même possible d'utiliser des pointeurs pointant sur des fonctions. En langage C, une fonction ne peut être considérée comme une variable, mais il est possible de définir "une variable qui pointe sur une fonction", celle-ci peut être manipulée, placée dans un tableau ...et même transmise à d'autres fonctions.
L'utilité des pointeurs est alors encore mise en évidence par la constatation de ce que le langage C n'autorise que le type de passage "par valeur" d'arguments à une fonction. Or on peut considérer que le fait de passer un pointeur à une fonction revient à passer la valeur de celui-ci à la fonction. Parce que précisément la valeur de ce pointeur est l'adresse de la donnée sur laquelle il pointe, c'est l'adresse qui est en fait passée. Il est ainsi devenu possible de passer à une fonction des arguments entiers de structures complexes, par adresse au lieu d'en passer une copie pénalisante en place mémoire et en temps, et même d'autres fonctions.

L'intégration des pointeurs et des tableaux est une des principales qualités du langage que complète la logique et la souplesse de C dans sa façon d'aborder l'allocation "dynamique" de mémoire.
Au lieu de gérer une zone mémoire déclarée de taille fixe et déjà compilée, C permet en effet au programmeur, par des appels aux fonctions "allocatrices de mémoires" de la bibliothèque, de solliciter le système d'exploitation pour obtenir plus de mémoire en cas de nécessité. Le stockage des espaces libres est gardé dans une chaîne de blocs reliés par des pointeurs. La chaîne est mise à jour à chaque occupation ou libération de zones "mémoire".

A l'origine ce langage a été développé principalement dans le but de faciliter l'écriture et la portabilité des systèmes d'exploitation et de leur environnement logiciel. Mais petit à petit il est apparu que les exigences des logiciels modernes le faisaient préférer à ses concurrents. Devant l'apparition de progiciels de plus en plus performants, tant au niveau de la vitesse d'exécution que des fonctions toujours plus nombreuses qu'ils offrent, seuls l'assembleur et le C sont capables de générer du code compact, avec une grande vitesse d'exécution. Mais l'assembleur est très spécifique à chaque machine (et à chaque micro-processeur) et il nécessite de très nombreuses lignes de codage. La mise au point de ces programmes est souvent longue et laborieuse. Tandis que le C autorise à la fois une programmation proche de la machine (mais sans dépendance directe), extrêmement rapide et performante et une programmation de haut niveau, de type structurée. C'est ce qui explique sa "réussite" et son emploi pour l'écriture de grands progiciels du marché.

Il a souvent été fait mention d'effet de mode à son sujet. Ce qui possède certainement une part de vérité mais il est difficile de soutenir la thèse d'une quelconque mode lorsque l'on s'aperçoit que les logiciels les plus performants l'ont employé avec profit. Il peut y avoir un effet de mode au niveau des amateurs éclairés en micro-informatique mais beaucoup moins en ce qui concerne les développeurs professionnels, soucieux avant toute autre chose ... de productivité et d'efficacité.

* — * — *

Chapitre 2 :
Le chapitre statistique

L'appellation d'"analyse des données" recouvre différents groupes de méthodes desquelles nous distinguons les méthodes de classification automatique et les méthodes d'analyse discriminante.

1 Classification automatique

La classification automatique porte sur une population, ensemble d'individus caractérisés par un certain nombre de variables, de laquelle il convient de discerner une partition (c'est-à-dire une famille de "sous-populations", appelées classes, non vides telle que chaque individu appartient à une et une seule de ces classes, dont l'union comprend l'ensemble complet de départ).
Les méthodes de classification automatique forment un ensemble ... "un peu difficile à classer". Il existe, en effet, autant de façons de réaliser "simplement" une classification qu'il existe de formes d'intuition, et donc de définitions de la simplicité. La classification automatique a dès lors particulièrement éveillé l'imagination des statisticiens. On rencontre ainsi dans la littérature plusieurs définitions du terme "classe" qui restent toutefois assez vagues, mais traduisent dans tous les cas une idée de "cohésion interne" et d'"isolation externe". A savoir que d'une part, les individus d'une classe ont certaines caractéristiques communes qui les distinguent du reste de la population, et d'autre part que tout individu d'une classe est plus "proche" de chaque individu de "sa" classe que de tout autre. La manière dont on tient compte des variables qui caractérisent les individus pour établir cette "proximité" et donc le partitionnement induit le type de la méthode.

Les critères utilisés dans la plupart des méthodes de classification furent souvent basés sur le calcul d'une (dis-)similarité intra classes, traduite par une notion de distance; la métrique utilisée définissant plus particulièrement la méthode.

Le développement du calcul géométrique sur ordinateur et en particulier la mise au point d'algorithmes de calcul de l'enveloppe convexe de plus en plus performants ont amené Rasson et Hardy à élaborer une méthode non pas en termes de dissimilarités, comme il est de coutume, mais utilisant les propriétés de la mesure naturelle de l'espace, à savoir la mesure de Lebesgue. La méthode définit succintement le problème de classification automatique comme suit : disposant d'un échantillon d'individus représentés chacun par un point d'un espace euclidien multidimensionnel, point dont les coordonnées "correspondent" aux valeurs prises par les variables caractérisant l'individu, il s'agit de trouver une

partition de cet échantillon en k classes dont les enveloppes convexes sont disjointes et dont la somme de leurs mesures de Lebesgue est minimale. Le modèle développé considère l'échantillon à classifier comme étant une réalisation d'un processus de Poisson homogène dans l'union C de k domaines convexes compacts disjoints C_p $(p = 1 \rightarrow k)$; chaque domaine correspond à une classe.

Rasson et Hardy ont associé à ce modèle une règle de "classification" que nous ne développerons pas ici afin de ne pas alourdir notre propos. Le lecteur désireux de s'informer sur ce sujet peut consulter Hardy, A. et Rasson, J.-P., "Une nouvelle approche des problèmes de classification automatique". Statistique et analyse des données (7), 1982 et Hardy, A. "Statistique et classification automatique : un modèle, un nouveau critère et des algorithmes: applications". PhD Thesis, Facultés Universitaires de Namur, Belgique, 1983.

* — *

2 Analyse discriminante et problème de classement

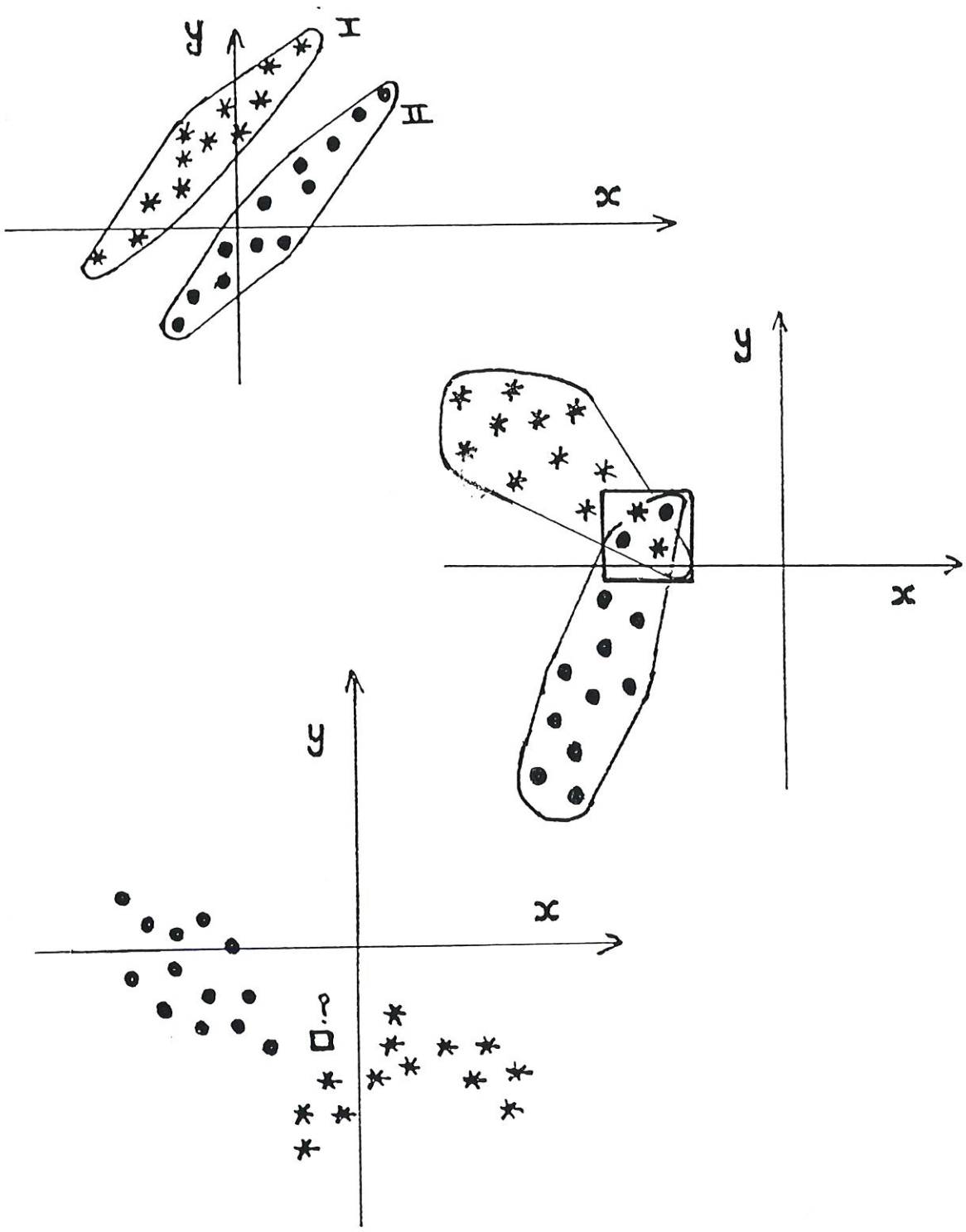
L'analyse discriminante est une méthode de description et d'estimation extrêmement puissante dont le champ des applications possibles est vaste.

2.1 Diagnostic médical et discrimination

Considérons une population, par exemple, de personnes souffrant d'une maladie cardiaque. Dans un premier temps, on observe sur ces malades un ensemble de symptômes caractéristiques. Chaque malade peut être représenté par un point dans l'espace défini par les symptômes. Il est ensuite examiné par un médecin qui pose sur son cas un diagnostic. On peut ainsi scinder le "nuage" des malades, repérés dans l'espace des symptômes, en plusieurs sous-"nuages" relatifs chacun à un diagnostic.

Notre perception nous interdit de "voir" quoi que ce soit dans un espace dont la dimension est supérieure à trois : un tel espace est une construction logique, qui peut obéir à des conventions semblables aux règles que nous expérimentons dans nos espaces usuels à 2 ou 3 dimensions mais qui ne peut pas être représenté par notre imagination. Illustrons ainsi (figure 1) un cas où l'on observe dès lors deux symptômes x et y et où le nombre de diagnostics différents est également deux.

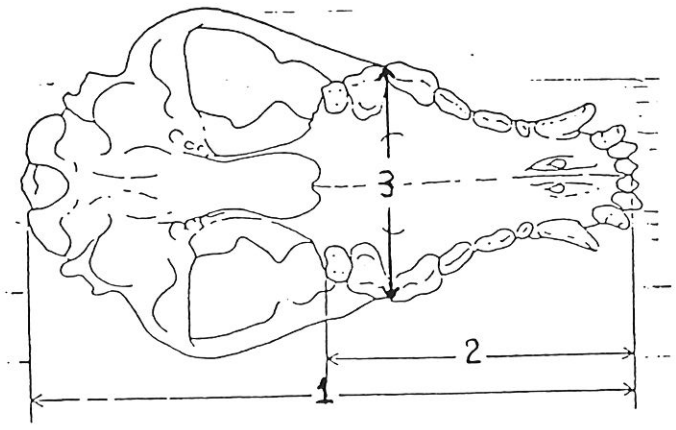
Le symptôme x à lui tout seul ne permet pas de discriminer parfaitement les diagnostics I et II : à la plupart des valeurs de x peuvent correspondre les deux diagnostics. Il en est de même de y. Par contre, il est facile de voir que $z = (x - y)$ permet de discriminer parfaitement les diagnostics I et II.
L'analyse discriminante se propose principalement primo de mettre à jour cette combinaison des symptômes ((mal-)heureusement pas toujours aussi "triviale") qui permet de discriminer au mieux les diagnostics, secundo de repérer des zones d'incertitude, dans lesquelles les individus peuvent a priori recevoir plusieurs diagnostics différents (figure 2), et enfin tertio, considérant un individu supplémentaire pour lequel on connaît les valeurs prises par les symptômes mais non le diagnostic, de lui attribuer un diagnostic (figure 3) : on observera comment l'individu se situe dans l'espace par rapport aux sous-nuages, et on lui attribuera le diagnostic qui correspond au sous-nuage dont il est le plus "proche".


y
I
II
x
y
x
y
x
?

La manière de définir la notion de proximité détermine une procédure qui permet de passer automatiquement des symptômes observés à un diagnostic, une fois la méthode étalonnée sur un nombre suffisant d'observations. Certes, la métaphore médicale n'est guère soutenable, car il serait dangereux d'évaluer automatiquement des diagnostics. Mais le procédé peut être utilisé dans bien d'autres domaines où il se révèle extrêmement "bien à propos".

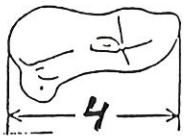
2.2 Identification paléontologique et classement

Illustrons un cas "heureux" de l'utilisation de l'analyse discriminante pour le "classement" d'un individu supplémentaire, à travers un exemple concernant l'identification d'un crâne fossile (paléontologie).
L'objet de l'étude est la comparaison d'un crâne de canidé provenant d'un niveau récent du quaternaire avec deux populations composées de 30 crânes de chiens domestiques de haute taille, pour l'une, et, pour l'autre, de 12 crânes de loups.
Préalablement à cette opération on précise la position relative des deux populations de crânes de chiens et de loups en tenant compte de caractéristiques sélectionnées.
Soit I l'ensemble des crânes (chiens domestiques, loups et canidé "supplémentaire"), et J l'ensemble des caractéristiques mesurées sur ces crânes : LCB, LMS, LBM, LC, LM et LAM telles que (schéma des mensurations relevées) $\cdots$
$\cdots$ sur le calvarium en vue ventrale :



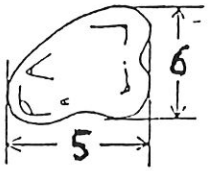
1 (LCB) : longueur condylo-basale.
2 (LMS) : longueur de la machoire supérieure.
3 (LBM) : longueur bi-maxillaire.

... sur la carnassière supérieure gauche en vue occlusale :



4 (LC) : longueur de la carnassière.

... sur la première molaire supérieure gauche en vue occlusale :



5 (LM) : longueur de la molaire.
6 (LAM) : largeur de la molaire.

Soit $k(i,j)$ la mesure de la caractéristique j sur le crâne i effectuée en mm, alors...
...Tableau k_{IJ} des mensurations effectuées :

		LCB	LMS	LBM	LC	LM	LAM
?	Canidé ?	210	103	72	20.5	14.0	16.7
•	Loup	199	105	73	23.4	15.0	19.1
•	Loup	227	117	77	25.0	15.3	18.6
•	Loup	228	122	82	24.7	15.0	18.5
•	Loup	232	123	83	25.3	16.8	15.5
•	Loup	231	121	78	23.5	16.5	19.6
•	Loup	215	118	76	25.7	15.7	19.0
•	Loup	184	100	69	23.3	15.8	19.7
•	Loup	175	94	73	22.2	14.8	17.0
•	Loup	239	124	77	25.0	16.8	27.0
•	Loup	203	109	70	23.3	15.0	18.7
•	Loup	226	118	72	26.0	16.0	19.4
•	Loup	226	119	77	26.5	16.8	19.3
*	Bull-dog	129	64	95	17.5	11.2	13.8
*	Bull-dog	154	74	76	20.0	14.2	16.5
*	Chien-ind	170	87	71	17.9	12.3	15.9
*	Chien-ind	188	94	73	19.5	13.3	14.8
*	Chien-ind	161	81	55	17.1	12.1	13.0
*	Chien-ind	164	90	58	17.5	12.7	14.7
*	Berger allemand	203	109	65	20.7	14.0	16.8
*	Lévrier	178	97	57	17.3	12.8	14.3
*	Lévrier	212	114	65	20.5	14.3	15.5
*	Lévrier	221	123	62	21.2	15.2	17.0
*	Colley	183	97	52	19.3	12.9	13.5
*	Doberman	212	112	65	19.7	14.2	16.0
*	Setter	220	117	70	19.8	14.3	15.6
*	Pyrénées	216	113	72	20.5	14.4	17.7
*	Bas-rouge	216	112	75	19.6	14.0	16.4
*	Berger allemand	205	110	68	20.8	14.1	16.4
*	Briard	228	122	78	22.5	14.2	17.8
*	Lévrier	218	112	65	20.3	13.9	17.0
*	Bull-mastif	190	93	78	19.7	13.2	14.0
*	Briard	212	111	73	20.5	13.7	16.6
*	Setter	201	105	70	19.8	14.3	15.9
*	Setter	196	106	67	18.5	12.6	14.2
*	Boxer	158	71	71	16.7	12.5	13.3
*	Dogue allemand	255	126	86	21.4	15.0	18.0
*	Dogue allemand	234	113	83	21.3	14.8	17.0
*	Gronendal	205	105	70	19.0	12.4	14.9
*	Berger allemand	186	97	62	19.0	13.2	14.2
*	Dogue allemand	241	119	87	21.0	14.7	18.3
*	Saint-Bernard	220	111	88	22.5	15.4	18.0
*	Dogue allemand	242	120	85	19.9	15.3	17.6

L'utilisation de l'analyse discriminante pour (tenter d') apporter une solution à ce problème d'identification paléontologique apparaît immédiatement bien fondée, comme il peut l'apparaître en bien d'autres circonstances. Mais dire ou penser n'est pas faire; pour apporter une réponse à laquelle "on" puisse croire, il est nécessaire que nous nous donnions un formalisme complet du problème général de "classement" et de la manière de l'aborder.

* — *

3 Formalisation du problème de "classement"

Considérons un échantillon E de N individus d'une population $\mathcal{P}$ sur lesquels nous avons mesuré M variables quantitatives X_m $(m = 1 \rightarrow M)$. Les individus sont repérés par les indices n $(n = 1 \rightarrow N)$.
Supposons maintenant que nous considérons une variable qualitative Y, telle qu'à chaque individu corresponde une modalité de Y et une seule. Nous noterons K le nombre de modalités de Y, et nous les repérerons par un indice k $(k = 1 \rightarrow K)$ lorsque nous considérerons une classe particulière et par des indices i et j lorsque ces classes seront considérées deux à deux.
Y détermine une partition (E_k) de l'"ensemble de référence" E en sous-populations.
Notons N_k le nombre des individus ayant la modalité k de Y, de sorte que $N_k = \sharp E_k$.

Supposons alors que nous considérions un individu supplémentaire de $\mathcal{P}$ pour lequel nous connaissons les valeurs des variables X_m mais non la modalité du caractère Y.
Il s'agit de "classer" cet individu supplémentaire dans un et un seul des K groupes, c'est-à-dire d'"estimer" la modalité de Y à associer à cet individu n^0 $(N+1)$ (ou 0 pour alléger la notation), sur base du profil de ses caractéristiques c'est-à-dire sur base des valeurs prises par les M variables X_{0m}, dites explicatives. La variable Y_0 est dite à expliquer.

Les variables X_m ont été définies comme quantitatives. Nous pouvons donc associer à chaque individu, un vecteur x d'un sous-espace V de $\mathbb{R}^M$ dont les M coordonnées sont les valeurs x_j prises par X_m $(m = 1 \rightarrow M)$, en particulier nous associons le vecteur $x_0 = (x_{0m})_{m=1 \rightarrow M}$ à l'individu "zéro" à classer.

$$\text{individu} \leftrightarrow x = (x_m)_{m=1 \rightarrow M}$$

Ce que nous souhaitons, c'est pouvoir attribuer à tout élément de V, donc à tout individu, une parmi les K classes de référence. Cette "attribution" revient à définir l'application d suivante, que l'on appelle règle de décision :

$$d : V \rightarrow \{k : 1 \rightarrow K\} : x \rightsquigarrow d(x)$$

Cette règle dépend des individus composant l'ensemble, et plus précisément les classes, de référence (ne serait-ce que parce que l'appartenance des individus de E à leurs groupes respectifs n'est, quelque soit l'origine de cette "information", jamais remise en question) et doit être définie de telle sorte qu'elle commette le moins d'erreurs de classement possible,

c'est-à-dire que la partition (V_k) qu'elle détermine sur V soit le plus proche possible de la réalité.

$$\forall\ k = 1 \rightarrow K\ :\ \ V_k = \{x \text{ tel que } d(x) = k\}$$

La *règle* d peut être définie par utilisation de K fonctions h_k dites discriminantes, associée chacune à une classe de référence. $\forall\ k = 1 \rightarrow K \quad h_k$, définie sur V, mesure la "ressemblance" $h_k(x)$ entre l'individu associé au vecteur x de V et les individus de la classe E_k.
Les ensembles $V.$ sont définis par ces fonctions $h.$ comme :

$$V_i = \{x \text{ tel que } h_i(x) \geq h_j(x)\ ,\ j = 1 \rightarrow K\} \qquad\qquad i = 1 \rightarrow K$$

Les ensembles D_{ij}, constituant les frontières entre les V_i et V_j sont appelées surfaces de décision. La présence de K classes implique l'existence de $K(K-1)/2$ surfaces de décision.

$$D_{ij} = \{x \text{ tel que } h_i(x) = h_j(x)\} \qquad\qquad i = 1 \rightarrow K\ ;\ j = 1 \rightarrow K\ ;\ j \neq i$$

On rencontre dans la littérature de nombreuses méthodes de discrimination. Selon que celles-ci utilisent ou (resp.) non des hypothèses ou outils statistiques, on parle de méthodes statistiques ou (resp.) déterministes.
Les méthodes statistiques, sur lesquelles seulement nous porterons notre attention, supposent que tout individu de $i^{\text{ème}}$ catégorie $(i = 1 \rightarrow K)$ est une réalisation d'un vecteur aléatoire de R^M, de densité $f_i(x)$.

Une erreur, entachant le classement, consiste à affecter un individu du groupe i $(i = 1 \rightarrow K)$ au groupe j $(j = toK$ et $j \neq i)$. La probabilité P_{ji} de commettre une erreur de ce type est donnée par

$$P_{ji} = \int_{V_j} f_i(x)dx \qquad\qquad j = 1 \rightarrow K\ \ ;\ \ i = 1 \rightarrow K\ \ ;\ \ i \neq j$$

En théorie, deux ou plusieurs différentes de ces erreurs pourraient ne pas avoir le même "poids". Pour tenir compte de ces "gravités" éventuellement différentes, il faut associer au classement d'un individu du groupe i dans le groupe j, un coût c_{ji}. Supposons que $c_{ii} = 0$ de sorte qu'aucune bonification n'est accordée à un classement correct, disposant d'une règle d, à chaque "classe" i $(i = 1 \rightarrow K)$ nous pouvons associer le risque $R_i(d)$ défini par :

$$R_i(d) = \sum_{j=1}^{K} c_{ji}P_{ji} \qquad\qquad i = 1 \rightarrow K$$

La règle "idéale" serait sans conteste celle qui annule tous les risques $R_i(d)$. Cette opportunité n'est malheureusement que trop rare.

Notre but consistera donc en la recherche d'une bonne règle, c'est-à-dire de l'une parmi les meilleures des règles "imparfaites" restantes, voire de la meilleure.
Le choix de la "meilleure" règle n'est pas aisé.

Une règle d est dite dominante par rapport à une règle d' si :
d'une part : $\forall\, i = 1 \rightarrow K : R_i(d) \leq R_i(d')$
et d'autre part : $\exists j\ ;\ 1 \leq j \leq K$ tel que $R_j(d) < R_j(d')$

Une règle qui n'est dominée par aucune autre est dite admissible.

Pour chaque règle non admissible, on peut trouver une règle dont le risque associé à chaque population est moindre : il convient donc de rechercher la "meilleure" règle parmi les règles admissibles.

Un complément d'information sur les probabilités a priori p_i d'appartenance d'un individu à la classe i $(1 = 1 \rightarrow K)$ (dans le cas le plus simple $p_i = \sharp$ classe $i/\sharp$ ensemble E de référence $= N_i/N$) fournit un critère selon lequel il convient de choisir la règle qui minimise l'espérance du risque, appelée risque bayesien que l'on note $R(d)$ s'exprimant par :

$$R(d) = \sum_{i=1}^{K} p_i R_i(d)$$

où nous savons que :

$$R_i(d) = \sum_{j=1}^{K} c_{ji} P_{ji} \qquad\qquad i = 1 \rightarrow K$$

et

$$P_{ji} = \int_{V_j} f_i(x)dx \qquad\qquad j = 1 \rightarrow K\ ;\ i = 1 \rightarrow K\ ;\ i \neq j$$

de sorte que

$$R(d) = \sum_{i=1}^{K} p_i \sum_{j=1}^{K} c_{ji} \int_{V_j} f_i(x)dx$$

c'est-à-dire

$$R(d) = \sum_{j=1}^{K} \int_{V_j} \left[\sum_{i=1}^{K} p_i c_{ji} f_i(x)\right] dx$$

Pour minimiser le risque bayesien $R(d)$, il faut choisir

$$V_j = \{x \text{ tel que } \sum_{i=1}^{K} p_i c_{ji} f_i(x) \leq \sum_{j=1}^{K} p_j c_{ij} f_j(x) \ , \ i = 1 \rightarrow K\} \quad j = 1 \rightarrow K$$

ce qui nous permet de dire que la règle issue du critère, dite bayesienne, possède comme fonctions discriminantes

$$h_i(x) = -\sum_{j=1}^{K} p_j c_{ij} f_j(x) \qquad\qquad i = 1 \rightarrow K$$

Or ce qui nous intéresse, ce n'est pas la valeur exacte des fonctions $h_i(x)$, mais une comparaison entre elles, à partir de laquelle sont définis les V_i.
Nous allons donc essayer de simplifier au maximum la formulation des $h_i(x)$ de manière à obtenir une définition des V_i $(i = 1 \rightarrow K)$ "la plus simple".

Dans la pratique, les erreurs d'affectation sont généralement considérées comme "aussi" graves les unes que les autres. Nous appellerons c l'unique coût communément associé; les fonctions $h_i(x)$ s'écrivent alors :

$$h_i(x) = [-c \sum_{j=1}^{K} p_j f_j(x)] + [c_\bullet \, p_i f_i(x)] \qquad\qquad i = 1 \rightarrow K$$

Dans la comparaison des $h_i(x)$, le premier terme, constant, peut "tomber", ainsi que la constante multiplicative c du second terme. Il reste comme expression "nécessaire et suffisante" des fonctions discriminantes :

$$h_i(x) = p_i f_i(x) \qquad\qquad i = 1 \rightarrow K$$

Les ensembles V_i sont dès lors exprimés par :

$$V_i = \{x \text{ tel que } p_i f_i(x) \geq p_j f_j(x) \ , \ j = 1 \rightarrow K\} \quad i = 1 \rightarrow K$$

et nous pouvons définir la règle bayésienne, déterminant cette partition de V, de la façon suivante :

$$d : x \rightsquigarrow d(x) = \min\{i \text{ tel que } p_i f_i(x) \geq p_j f_j(x) \ , \ j = 1 \rightarrow K\}$$

La règle bayésienne consiste, en fait, à affecter un individu x au groupe qui a la probabilité a posteriori maximale de contenir cet individu. Cette probabilité est donnée par la formule de Bayes :

$$P(V_i|x) = \frac{p_i f_i(x)}{\sum_{j=1}^{K} p_j f_j(x)}$$

Les dénominateurs étant les mêmes pour les K groupes, il s'agit bien de chercher le maximum des $p_i f_i(x)$.

$$?\ \ i \text{ tel que } p_i f_i(x) = \max_{j \in \{1,\cdots,K\}} p_j f_j(x)$$

Le problème de classement est entièrement résolu, lorsque les densités $f_i(x)$, relatives à chaque catégorie, sont connues; ce qui n'est pas le cas dans la pratique où la seule information dont nous disposons à propos de chaque groupe V_i d'affectation à la classe i, est un échantillon E_i de taille finie N_i de celui-ci. Les densités $f_i(x)$, inconnues, doivent donc être estimées sur base de ces échantillons.

Les méhodes statistiques se séparent alors en deux catégories, les méthodes paramétriques, où l'on impose une forme paramétrique à ces densités (se donnant une famille paramétrée de lois de probabilités pour $f_i(x)$, on utilise l'échantillon pour estimer les paramètres), et les méthodes non paramétriques, où on ne fait aucune hypothèse spécifique à propos des densités. La suite de l'exposé s'inscrit dans cette dernière option.

* — *

4 Classement sous l'hypothèse d'une réalisation ··· ··· / ··· d'un processus de Poisson homogène

Les propriétés, tant théoriques que pratiques, remarquables de la méthode de classification développée par Rasson et Hardy ont séduit Baufays dont les travaux (Baufays, P. et Rasson, J.-P. "Une nouvelle règle de classement utilisant l'enveloppe convexe et la mesure de Lebesgue", Statistique et analyse des données 9, 1984 et Baufays, P. "Une nouvelle règle géométrique en analyse discriminante : critère de classement, affectations, courbes de décision et applications réelles", PhD Thesis, Facultés Universitaires de Namur, Belgique, 1985) poursuivent la recherche de Rasson et Hardy en utilisant le même modèle statistique en analyse discriminante.

Par soucis de légèreté nous n'exposerons que l'essentiel du raisonnement suivi par Baufays pour développer sa règle de classement. (Le lecteur pourra se référer aux publications citées ci-dessus). Sous les hypothèses de ce modèle ···

Pour rappel : le modèle considère l'échantillon à "classifier" comme étant une réalisation d'un processus de Poisson homogène dans l'union C de K domaines convexes compacts C_k $(k = 1 \rightarrow K)$; chaque domaine correspond à une classe.

... d'une part la probabilité p_i d'appartenance d'un individu à la classe i est proportionnelle à →

··· d'autre part tout individu de la $i^{\text{ème}}$ population satisfait une distribution uniforme dans C_i, dont la densité conditionnelle $f_i(x)$ sachant que x est de $i^{\text{ème}}$ catégorie dépend de →

↣ la mesure de Lebesgue m des enveloppes convexes $H(E_1), H(E_2), \cdots$ et (resp.) $H(E_K)$ des ensembles de référence E_1, (resp.) $E_2, \cdots$ et E_K.

$$\cdot\ p_i = \frac{m(C_i)}{\sum_{j=1}^{K} m(C_j)}$$

$$f_i(x) = \frac{1}{m(C_i)}\ \mathbb{1}_{C_i}(x)$$

où $\mathbb{1}_{C_i}(x)$ est l'indicatrice du domaine $C_i \cdots$
et où les domaines C_j, inconnus, sont estimés de la manière suivante :

Si l'individu à classer x_0 est affecté au $i^{\text{ème}}$ groupe, alors $\cdots$

$$C_i = H(E_i \cup \{x_0\})$$
$$\text{et} \quad C_j = H(E_j) \quad j = 1 \rightarrow K \ ; \ j \neq i$$

4.1 Premier critère : cas disjoints

La règle de classement définie par la recherche du maximum des $p_i f_i(x)$ n'est, dans un premier temps, applicable que lorsque ces enveloppes convexes sont disjointes.

L'individu x_0 à classer est affecté à la classe i telle que la mesure de Lebesgue ajoutée par convexité, c'est-à-dire que x_0 "ajoute" à l'enveloppe convexe de E_i lorsqu'il y est incorporé, soit minimale et telle que l'enveloppe convexe issue de cette incorporation n'intersecte pas avec les enveloppes convexes des classes restantes.

C'est-à-dire i tel que $S_i(x) = \min\{S_j(x)\ 1 \leq j \leq K\}$
où $S_j(x) = m(H(E_j \cup \{x_0\})) - m(H(E_j))$
Si $\forall\ k = 1 \rightarrow K\ ,\ k \neq j\ : H(E_j \cup \{x_0\}) \cap H(E_k) = \phi$
$= +\infty$ sinon.

En particulier un individu appartenant à l'une des enveloppes convexes est classé dans le groupe correspondant.

Le caractère disjoint des enveloppes convexes est une propriété désirable des ensembles de référence. Si ceux-ci proviennent d'une méthode de classification, cela devrait être le cas. Malheureusement, il arrive qu'il n'en soit pas ainsi.

4.2 Second critère : cas non disjoints

Afin d'assurer la plus grande applicabilité possible, dans un second temps, Baufays décrit un critère de classement associé au même modèle amputé cependant de l'hypothèse de disjonction des enveloppes convexes des ensembles de référence.

Dans ce cas, le critère distingue les éventualités suivantes :

Si l'individu à classer x_0 n'appartient à aucune enveloppe convexe, il est classé dans le groupe i tel que $S_i(x)$ est minimal parmi les $S_j(x)$ $j : 1 \rightarrow K$.
En particulier ...
...Si x_0 appartient à une et une seule enveloppe convexe $H(E_i), x_0$ est affecté à la $i^{\text{ème}}$ catégorie.

Si x_0 appartient à p $(1 \leq p \leq K)$ enveloppes convexes, soit $H(E_{i1}), \cdots, H(E_{ip})$, plutôt que de classer x_0 de manière arbitraire dans l'un des groupes $i_1, \cdots, i_p$, Baufays conseille de ne rien faire et de considérer l'individu comme inclassable.

La démarche basée sur le processus de Poisson homogène ne permet donc pas de classer des points appartenant à l'intersection des enveloppes convexes de plusieurs ensembles de référence.

* — *

5 Classement sous l'hypothèse d'une réalisation ··· ···/··· d'un processus de Poisson non homogène

Dans le but d'apporter, malgré tout, une solution au "classement" de ces points "inclassables" (du moins considéés comme tel par Baufays) Rasson et Granville (cfr. Ochao, S. "Une nouvelle méthode de "classification" basée sur le processus de Poisson non-homogène", Mémoire, Facultés Universitaires de Namur, Belgique, 1992) généralise l'hypothèse du processus de Poisson homogène à celle d'un processus de Poisson non homogène.

Le modèle établi considère l'échantillon à "classifier" comme étant une réalisation d'un processus de Poisson non-homogène, d'intensité, inconnue, $q(x)$ strictement positive, dans l'union D de K domaines convexes D_k $(k = 1 \rightarrow K)$; chaque domaine correspond à une classe.

Sous les hypothèses de ce modèle, tout individu x satisfait à une distribution dans D dont la densité $f_D(x)$ s'exprime par

$$f_D(x) = \frac{q(x)\mathbb{1}_D(x)}{\int_D q(x)dx}$$

Notons, $\forall\, i = 1 \rightarrow K$, $q_i(x)$ la restriction de q à D_i c'est-à-dire :

$$q_i(x) = q(x)\mathbb{1}_{D_i}(x) \quad i = 1 \rightarrow K\ ;$$

nous obtenons une décomposition de $f_D(x)$ en une somme de $p_i f_i(x)$,

$$f_D(x) = \sum_{i=1}^{K} p_i f_i(x)$$

où p_i est la probabilité d'appartenance d'un individu à la classe i

$$p_i = \frac{\int_{D_i} q_i(x)dx}{\int_D q(x)dx} \qquad\qquad i = 1 \rightarrow K$$

et $f_i(x)$ est la densité conditionnelle, sachant que x est de $i^{\text{ème}}$ catégorie, de la distribution dans D_i à laquelle satisfait tout individu de la $i^{\text{ème}}$ population.

$$f_i(x) = \frac{q_i(x)}{\int_{D_i} q_i(x)dx} \qquad\qquad i = 1 \rightarrow K$$

Les paramètres inconnus de la $i^{\text{ème}}$ population sont le domaine convexe D_i, lui-même, d'une part et $q_i(x)$, restriction à D_i de la densité inconnue, $q(x)$, du processus.

L'estimation d'un convexe, connue au travers d'une réalisation dans celui-ci d'un processus de Poisson non homogène, est traitée par Ochao; pour notre propos il suffit de rappeler que l'estimateur du maximum de vraissemblance des domaines D_j est donné par :

$$\begin{aligned} D_j &= H(E_j) \qquad\qquad j = 1 \rightarrow K \;\; ; \;\; j \neq i \\ \text{et } D_i &= H(E_i \cup \{x_0\}) \text{ si l'individu à classer } x_0 \text{ est affecté au } i^{\text{ème}} groupe. \end{aligned}$$

5.1 Premier critère : cas disjoints

La règle de classement est définie par la recherche du maximum des $p_i f_i(x_0)$.

L'individu x_0 à classer est affecté à la classe i telle que $\cdots$

$$p_i f_i(x_0) \geq p_j f_j(x_0) \quad \forall\, j = 1 \rightarrow K$$

c'est-à-dire :

$$\frac{q_i(x_0)}{\int_D q(x)dx} \geq \frac{q_j(x_0)}{\int_D q(x)dx}$$

ou encore :

$$\frac{q_i(x_0)}{\sum_{p=1}^{K} \int_{D_p} q_{(p)}(x)dx} \geq \frac{q_j(x_0)}{\sum_{p=1}^{K} \int_{D_p} q_{(p)}(x)dx}$$

où les domaines D_p sont estimés par $\rightarrow$

$\rightarrow$ dans le terme de gauche (l'individu x_0 est affecté au $i^{\text{ème}}$ groupe) :

$$\begin{aligned} D_p^{(i)} &= H(E_p \cup \{x_0\}) \quad p = i \\ D_p^{(i)} &= H(E_p) \qquad\qquad p \neq i \end{aligned}$$

$\rightarrow$ dans le terme de droite (l'individu x_0 est affecté au $j^{\text{ème}}$ groupe) :

$$\begin{aligned} D_p^{(j)} &= H(E_p \cup \{x_0\}) \quad p = j \\ D_p^{(j)} &= H(E_p) \qquad\qquad p \neq j \end{aligned}$$

La règle s'écrit : l'individu x_0 à classer est affecté à la classe i tel que ...

$$\frac{q_i(x_0)}{\sum_{p=1}^{K} \int_{D_p^{(i)}} q_p(x)dx} \geq \frac{q_j(x_0)}{\sum_{p=1}^{K} \int_{D_p^{(j)}} q_p(x)dx} \qquad j = 1 \rightarrow K$$

Ou encore, si $q_i(x_0) = q_j(x_0)$, tel que...

$$\sum_{p=1}^{K} \int_{D_p^{(i)}} q_p(x)dx \leq \sum_{p=1}^{K} \int_{D_p^{(j)}} q_p(x)dx \qquad j = 1 \rightarrow K$$

C'est-à-dire :

$$\sum_{p=1_{p\neq i}}^{K} \int_{H(E_p)} q_p(x)dx + \int_{H(E_i \cup \{x_0\})} q_i(x)dx \leq \cdots / \cdots$$

$$\cdots / \cdots (\leq) \sum_{p=1_{p\neq j}}^{K} \int_{H(E_p)} q_p(x)dx + \int_{H(E_j \cup \{x_0\})} q_j(x)dx$$

$$\sum_{p=1}^{K} \int_{H(E_p)} q_p(x)dx + \int_{H(E_i \cup \{x_0\})} q_i(x)dx - \int_{H(E_i)} q_i(x)dx \leq \cdots / \cdots$$

$$\cdots / \cdots (\leq) \sum_{p=1}^{K} \int_{H(E_p)} q_p(x)dx + \int_{H(E_j \cup \{x_0\})} q_j(x)dx - \int_{H(E_j)} q_j(x)dx$$

Finalement : i tel que ...

$$\int_{H(E_i \cup \{x_0\})} dx - \int_{H(E_i)} q_i(x)dx \leq \int_{H(E_j \cup \{x_0\})} q_j(x)dx - \int_{H(E_j)} q_j(x)dx \qquad j = i \rightarrow K$$

Définissons $\forall\, j = 1 \rightarrow K$:

$$\begin{aligned} S_j(x_0) &= \int_{H(E_j \cup \{x_0\})} q_j(x)dx - \int_{H(E_j)} q_j(x)dx \\ &\text{si } \forall\, p = 1 \rightarrow K\ ,\ p \neq j\ :\ H(E_j \cup \{x_0\}) \cap H(E_p) = \phi \\ &= +\infty \\ &\text{sinon.} \end{aligned}$$

L'inégalité s'écrit alors : $S_i(x_0) \leq S_j(x_0)$ et la règle de classement consiste donc à affecter x_0 à la $i^{\text{ème}}$ catégorie tel que

$$S_i(x_0) \leq S_j(x_0) \quad j = 1 \rightarrow K$$

$S_j(x_0)$ est l'intensité que l'individu x_0 ajoute par convexité à la classe j, c'est-à-dire que x_0 "ajoute" à l'enveloppe convexe de E_j lorsqu'il y est incorporé.

En particulier, un individu appartenant à l'une des enveloppes convexes est classé dans le groupe correspondant.

$S_j(x)$ est définie ($= +\infty$ si $\cdots$) de façon à conserver lors de l'incorporation le caractère disjoint des enveloppes convexes $H(E_j)$ des populations E_j de référence.

Cette précaution est indispensable $\cdots$

Le modèle vis-à-vis duquel la théorie doit rester cohérente suppose disjoints les domaines D_i inconnus; il est donc indispensable de "tout faire" pour que les estimateurs que l'on se définit de ces domaines D_i, à savoir les enveloppes convexes des groupes de référence, soient également disjoints.

$\cdots$ mais impuissante lorsque dès le départ les enveloppes convexes $H(E_1), \cdots, H(E_K)$ des groupes de référence $E_1, \cdots, E_K$ sont d'elles-mêmes (indépendamment de l'incorporation dans telle ou telle classe de l'individu x_0) non disjointes.
Dans un tel cas le critère n'est plus applicable.

5.2 Second critère : cas non disjoints

Pour remédier à cette défectuosité, fortement restrictive dans les cas pratiques, un second critère de classement a été développé associé au modèle amputé de l'hypothèse selon laquelle les domaines D_i (et donc les enveloppes convexes ...) doivent être disjoints.

On distingue alors les éventualités suivantes :

Si l'individu à classer x_0 n'appartient à aucune enveloppe convexe, il est classé dans le groupe i tel que $S_i(x_0)$ est minimal parmi les $S_j(x_0)$ définis de manière commune par

$$S_j(x_0) = \int_{H(E_j \cup \{x_0\})} q_j(x)dx - \int_{H(E_j)} q_j(x)dx$$

En particulier, si x_0 appartient à une et une seule enveloppe convexe $H(E_i), x_0$ est affecté à la $i^{\text{ème}}$ catégorie.

Intéressons-nous au cas "critique" (pour lequel précisément, insistons là-dessus, le processus de Poisson homogène fut "laissé" pour le processus de Poisson non homogène) où l'individu à classer appartient à plusieurs enveloppes convexes, c'est-à-dire à l'intersection de p $(1 \leq p \leq K)$ domaines convexes D_i, soient $D_{i1}, \cdots, D_{ip}$.

La règle "originale" de classement définie précédemment reste applicable.
Pour rappel : L'individu x_0 à classer est affecté à la classe i telle que

$$\frac{q_i(x)}{\int_D q(x)dx} \geq \frac{q_j(x)}{\int_D q(x)dx} \qquad \forall\, j = 1 \rightarrow K$$

Cependant parce que x_0 ne peut être, en toute logique, affecté qu'à une des p classes $i_1, \cdots, i_p$ (classe = cohésion interne et isolement externe) et que donc x_0 appartient, quelque soit cette affectation $i.$, à l'enveloppe convexe $H(E_{i.})$ (par définition des $i_1, \cdots, i_p$), avant même son incorporation, l'affectation de x_0 n'ajoute, par convexité, aucune intensité à aucune classe; de sorte que les dénominateurs sont égaux dans l'inégalité.

Notre réflexion met donc en évidence une règle de classement pour le cas d'un point x_0 appartenant à l'intersection de plusieurs enveloppes convexes.

L'individu x_0 à classer est affecté à la classe $i.$ tel que

$$q_{i.}(x_0) \geq q_j(x_0) \quad \forall\, j \in \{i_1, \cdots, i_p\}$$

Les intensités $q_i(x)$ $(i = 1 \rightarrow K)$ sont cependant inconnues, il nous faut les estimer ...

Le lecteur trouvera chez Ochao un exposé de divers procédés d'estimation de $q(x)$.
Le procédé retenu consiste à estimer $q(x)$ par le "noyau de l'estimateur naïf appliqué au cas discret, proposé par Silverman (Silverman, B.W., "Density estimation for statistics and data analysis", School of mathematics, University of Bath, U.K.), c'est-à-dire $\cdots$

Considérons la réalisation $\{x_1, \cdots, x_N\} \cup \{x_0\}$ de notre processus ..., l'intensité $q(x)$, inconnue, du processus est estimée par :

$$\hat{q}(x) = \sum_{i=0}^{N} \mathcal{K}(x - x_i) \qquad x \neq x_i$$

En particulier ...

$$\hat{q}(x_0) = \sum_{i=1}^{N} \mathcal{K}(x_0 - x_i)$$

$$\text{où } \mathcal{K}(x_0 - x_i) \quad = \frac{1}{(2h+1)^M} \text{ si } |x_0 - x_i| \leq h \text{ et } 0 \text{ sinon}$$

de sorte que,

$$\hat{q}(x_0) = \frac{z}{(2h+1)^M}$$

où z est le nombre de points $x_i \quad (i = 1 \rightarrow N)$ qui satisfont : $|x_0 - x_i| \leq h \cdots\cdots$ ou encore $\forall\, j \;=\; 1 \rightarrow M \;:\; |x_{ij} - x_{0j}| \leq h$

Nous posant la question de la "connaissance opératoire" de z, nous apportons comme élément de réponse ...

Dans l'espace à M dimensions où est représenté chaque individu x, z représente le nombre de points, parmi $x_1, \cdots, x_N$, qui tombent dans une fenêtre, de largeur h, centrée en le point à classer x_0.
Nous disons que ces points recouvrent x_0, de sorte que par restriction de $\hat{q}(x_0)$ à chaque domaine $D_i \cdots$

$$\hat{q}_i(x_0) = \frac{z_i}{(2h+1)^M}$$

où z_i est le nombre de points de la classe E_i qui recouvrent x_0.
Considéran(l'égalité des dénominateurs, la règle de classement s'écrit finalement $\rightarrow$

$\rightarrow$ L'individu à classer x_0 est affecté $\cdots$
$\cdots$ à la classe i dont un maximum de points le recouvrent.
Ou encore serait-on tenté de dire ... à la classe i la plus "dense" autour $\cdots$ de x_0.

L'expression $(2h+1)^M$ est en effet dans l'espace discrétisé à M dimensions, l'expression de la mesure de la fenêtre (aire d'un pavé si $M = 2$, volume d'un cube si $M = 3$ et de façon générale hypervolume d'un hypercube si $M > 3$) placée autour de x_0 pour le calcul, ou plus exactement le "comptage", des z_i.
De sorte que les intensités $q_i(x_0)$ sont estimés par des $\hat{q}_i(x_0)$ qui sont exprimés comme le rapport d'un nombre de $\cdots$ par "volume" de $\cdots$ à ce volume, c'est-à-dire par une notion de "densité".

Le problème de clasement est entièrement résolu lorsqu'il est étali →

→ d'une part un critère pour le choix de la largeur h de la fenêtre;

Le "choix" du h est discuté par Ochao; pour notre propos il suffit de signaler que dans la plupart des cas pratiques, le "meilleur" h consiste en "la plus grande des plus petites maxi-distances", c'est-à-dire :

$$h = \max_{i \in \{1,\cdots,N\}} \min_{j \in \{1,\cdots,N\}_{j \neq i}} |x_i - x_j|$$

→ d'autre part, h étant supposé connu, un procédé efficace de comptage des nombres z_i de points de la classe de référence E_i qui recouvrent le point à classer x_0 ou encore peut-on dire un procédé efficace de mesure de la densité d'une classe autour d'un point.

* — * — *

Chapitre 3 :
Mesure de la densité d'une classe autour d'un individu

1 Le "champ" de la discussion

Considérons un échantillon E de N individus x_i $(i = 1 \rightarrow N)$ d'une population sur lesquels nous avons mesuré M variables quantitatives X_j $(j = 1 \rightarrow M)$.
Supposons maintenant que nous considérions une variable qualitative Y, telle qu'à chaque individu corresponde une modalité de Y et une seule. Nous notons K le nombre de modalités de Y et nous les repérons par un indice p $(p = 1 \rightarrow K)$.
Y détermine une partition (E_p) de l'ensemble de référence E en sous-populations ou classes de référence. Notons N_p le nombre des individus ayant la modalité p de Y.
Supposons alors que nous considérions un individu supplémentaire x_0 pour lequel nous connaissons les valeurs des variables X_j mais non la modalité du caractère Y. Il s'agit de classer cet individu supplémentaire dans une et une seule des K catégories ...

Le modèle développé considère l'ensemble des individus comme la réalisation $\{x_i \ (i = 1 \rightarrow N)\} \cup \{x_0\}$ d'un processus de Poisson non-homogène, d'intensité, inconnue, $q(x)$ strictement positive, dans l'union D de K domaines convexes D_p $(p = 1 \rightarrow K)$; chaque domaine correspond à une classe.

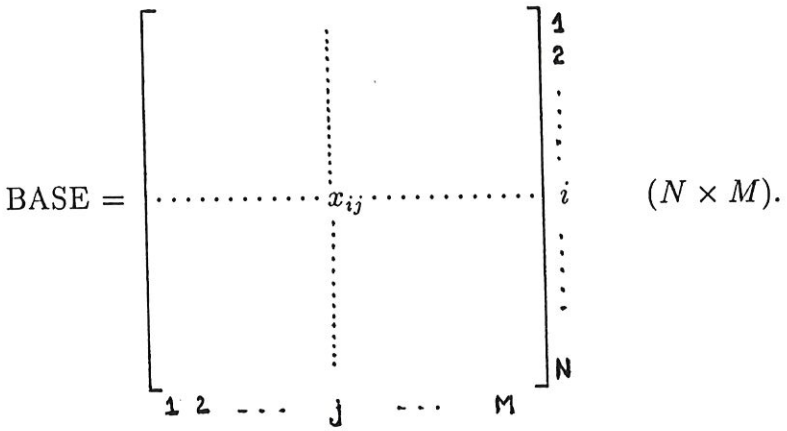
Sous les hypothèses de ce modèle, le processus de classement établi consiste à mesurer la densité de chacune des classes de référence E_p autour de l'individu x_0 et à affecter celui-ci à la classe dont cette mesure est maximale, le sens précis apporté à la notion de "mesure de densité" d'une classe autour d'un individu définissant plus particulièrement le procédé.

A savoir ... chaque individu x_i $(i = 1 \rightarrow N)$ est associé, dans une optique descriptive, à un point $x_i = (x_{i1}, \cdots, x_{iM})$ de l'espace à M dimensions, où x_{ij} est la valeur prise par la variable X_j sur l'individu x_i et $\forall\ p = 1 \rightarrow K$ la mesure de la "densité" de la classe E_p autour de l'individu x est amenée par : $\rho_p(x)$ = nombre de points de la classe E_p qui tombent dans une fenêtre de largeur h centrée en le point x.

* — *

2 Le problème

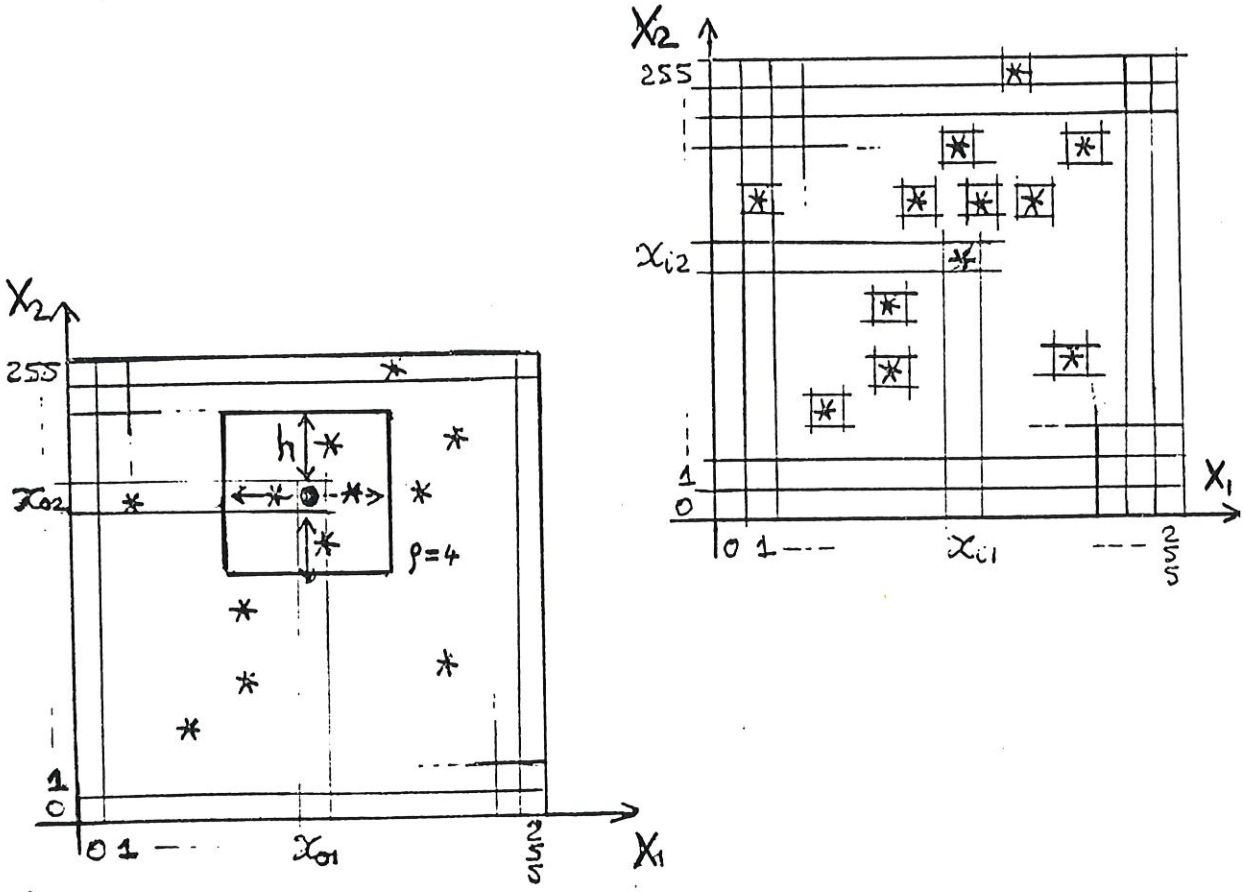
Soit E_p une quelconque de ces classes de référence $\cdots$
(Nous omettrons dans la suite de l'exposé, par soucis de légèreté, l'indice $p \in \{1, \cdots, K\}$).
$\cdots$ alors les observations des valeurs prises par les M variables X_j sur les N individus sont rassemblées en un tableau rectangulaire à N lignes et M colonnes que nous appelons BASE du fait de l'utilisation courante du terme anglais training-set (traduit par "base d'entraînement") pour désigner l'ensemble des points d'une classe de référence.



Chaque ligne $i : 1 \rightarrow N$ est identifiée à un individu de la classe E. c'est-à-dire à un point de l'espace à M dimensions , chaque colonne $j : 1 \rightarrow M$ à la variable X_j correspondante et x_{ij} est la valeur prise par la variable n$^0 j$ sur le i$^{\text{ème}}$ individu.

Supposons que les variables X_j (ne) puissent prendre (que) 256 valeurs discrètes c'est-à-dire que les x_i $(i = 1 \rightarrow N)$ soient des points du sous-espace $[0, 255]^M$ de l'espace discret à M dimensions $\mathbb{N}^M$; La justification de cette hypothèse se trouve essentiellement dans la pratique informatique que nous développerons plus loin dans l'exposé.
Il s'agit donc de compter le nombre $\rho(x_0)$ de points parmis les x_i qui tombent dans une fenêtre (pavé, cube, ..., hypercube) de largeur h centrée en le point $x_0 = (x_{01}, \cdots, x_{0M})$ associé à l'individu à classer. La connaissance de h n'est pas mise en question.

Nous visualisons le cas particulier où $M = 2$ variables sont mesurées sur $N = 12$ individus, au travers de la modélisation suivante : $\cdots$



Quelles méthodes, construites de manière générale, dans un espace à M dimensions, peuvent nous permettre de compter le nombre $\rho(x_0)$?
Nous en distinguons principalement trois que nous avons qualifié de naïve, projective et (resp.) naturelle, en fonction de l'"esprit" de chacune que nous développons ci-après $\cdots$

* — *

3 La méthode naïve

La méthode naïve consiste à visiter tous les points de la classe E et pour chacun de ces points effectuer le test $\cdots$ qui permet de répondre à la question : le point appartient-il (?), oui ou non, à la fenêtre considérée.

Il s'agit pour chacun des points x_i $(i = 1 \rightarrow N)$ d'effectuer le test suivant :

$$? \; \forall \, j : 1 \rightarrow M \quad |x_{ij} - x_{0j}| \leq h$$

D'un point de vue algorithmique cette méthode est la plus simple à implémenter :

```
| aux = 0
|
| pour i allant de 1 à N par pas de 1 faire :
|   | bool = 1
|   | "tester le point x_i"
|   | aux = aux + bool
|
| z = aux
```

Où "tester le point x_i" consiste en :

```
| j = 1
|
| tant que (bool = 1 et j ≤ M) faire :
|   | si |x_ij − x_0j > 1h
|   | alors bool = 0
|   | j = j + 1
```

Même si le test du point x_i n'est achevé que jusqu'à établissement de l'existence d'un "élément perturbateur" $\cdots$

$$\exists \; j \in \{1, \cdots, M\} \text{ tel que } |x_{ij} - x_{0j}| > h$$

$\cdots$ la méthode est horriblement fastidieuse dès que le nombre de points de la classe de référence, c'est-à-dire le nombre de test à effectuer, est important ou que les variables X_j les plus étroitement distribuées autour de x_{0j} sont les variables d'indices j premiers, de sorte que la plupart des tests sont effectués quasi intégralement.

Notre discussion porte donc essentiellement sur les deux méthodes restantes : à savoir la méthode projective et la méthode naturelle; ce à quoi nous nous attachons de suite $\cdots$

* — *

4 La méthode projective

4.1 Principe

La méthode projective est suggérée dès acte de projection (d'où son nom) du contenu de l'espace à M dimensions sur le plan principal des deux premières coordonnées, c'est-à-dire $[0,255]^M$ sur $[0,255] \times [0,255]$.
Les images des N points de référence $x_i = (x_{i1}, x_{i2}, \cdots, x_{iM})$ associés aux N individus de la classe E de référence, de même du point à classer $x_0 = (x_{01}, x_{02}, \cdots, x_{0M})$, par cette projection sont les points $\overline{x}_i = (x_{i1}, x_{i2})$, respectivement $\overline{x}_0 = (x_{01}, x_{02})$, de $[0,255]^2$.

Attention que, s'agissant d'une projection, plusieurs points x_j distincts peuvent avoir une seule et même image $\overline{x}$ par projection !
L'hypercube H, de mesure h, centré en x_0 est projeté, quant à lui, sur un pavé $\overline{H}$, de largeur h, centré en $\overline{x}_0$.

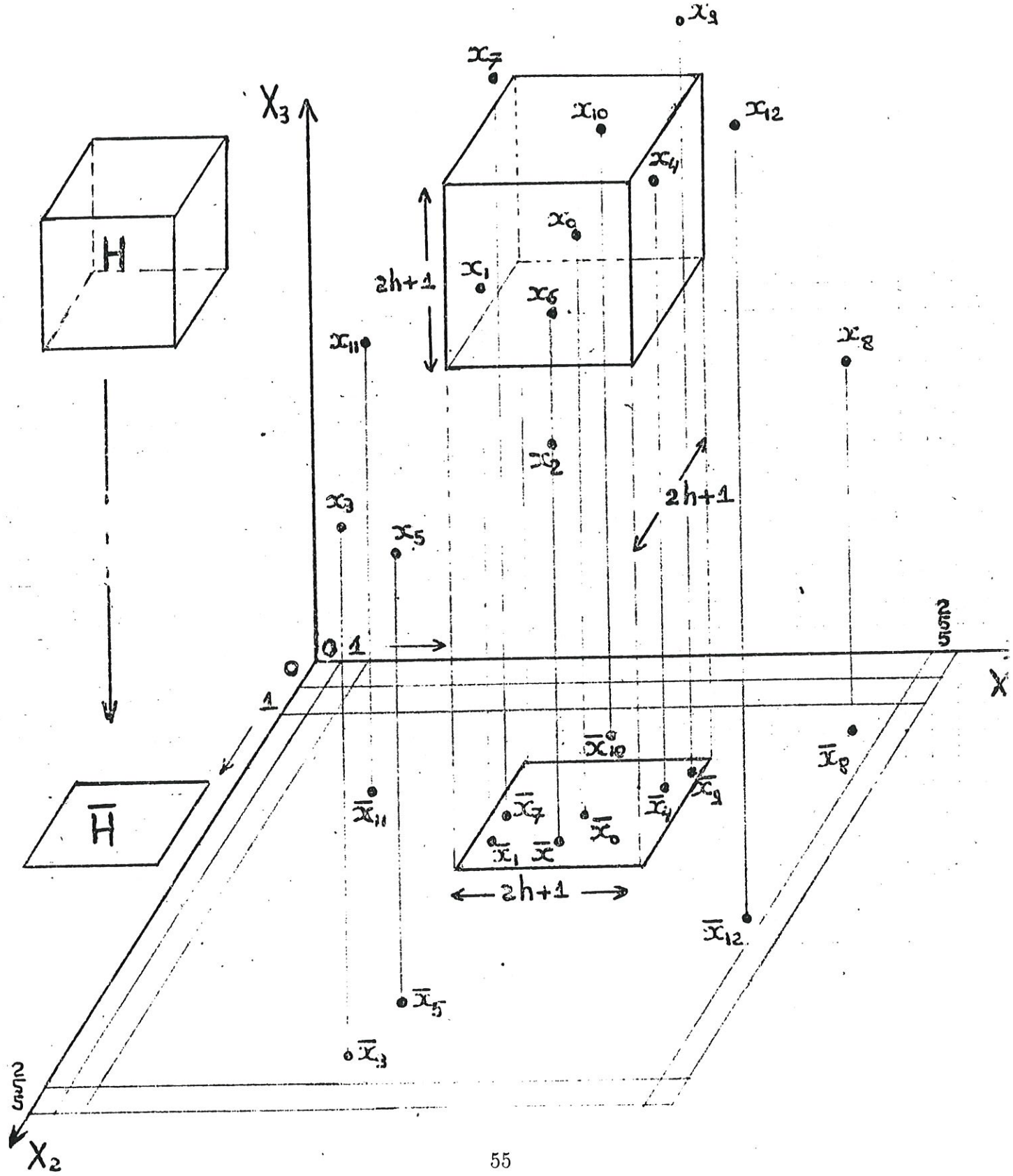
Le procédé est fondé sur la constatation du fait suivant : les points $x_i (\in E)$ de l'espace à M dimensions tombant dans l'hypercube H sont à rechercher parmis ceux dont les projections $\overline{x}_i$ tombent dans le pavé $\overline{H}$ de l'espace de dimension 2.

C'est-à-dire que : les points x_i dont l'image $\overline{x}_i$ par projection tombe dans $\overline{H}$ sont des points "suspects" pour le "comptage" des points x_i tombant dans H, lequel "comptage" est notre but.

4.2 Illustration

Visualisons le principe de la méthode sur un exemple où la classe de référence E contient 12 individus x_i et où il a été effectué sur chacun de ces individus (, sur les individus des autres classes de référence) et sur l'individu x_0 à classer un ensemble de trois mesures.
Le lecteur peut se reporter aux deux pages suivantes $\cdots$

Visualisation ($N = 12$ et $M = 3$) de la méthode projective.



Il s'agit de $\cdots$

$\cdots$ d'une part, repérer les points "suspects" (étape 1).

Les points x_i d'indice i=1, 2, 4, 6, 7 et (resp.) 9 sont tels que leurs images $\overline{x}_i$ par projection tombent dans le pavé $\overline{H}$; on remarquera que $\overline{x}_2 = \overline{x}_6 =$, soit, $\overline{x}$.

$\cdots$ d'autre part, parmis ces points "suspects", distinguer ceux qui effectivement (ne) tombent (pas) dans l'hypercube H (étape 2).

Les points x_i d'indice $i = 7$ et 9, trop en haut, et le point x_2, trop en bas, n'appartiennent pas à l'hypercube H : ils sont bien à la verticale de $\overline{H}$ mais pas pour autant, pourrait-on dire, à bonne altitude.

$$x_{73} > x_{03} + h \ , \ x_{93} > x_{03} + h \quad \text{et} \quad x_{23} < x_{03} - h$$

Les points x_i d'indice i=1, 4 et 6 sont les points qu'il convient de compter $\rightarrow \rho = 3$.

$$x_{03} - h \leq x_{13} \ , \ x_{43} \ , \ x_{63} \leq x_{03} + h$$

Le principe de la méthode projective est donc achevé en deux étapes qu'il convient maintenant de développer plus amplement.

4.3 Développement en deux étapes

4.3.1 Première étape : le repérage des points "suspects"

Formalisation

Les q points x_i "suspects" sont les points dont les deux premières coordonnées, x_{i1} et x_{i2}, sont telles que :

$$|x_{i1} - x_{01}| \leq h \quad \text{et} \quad |x_{i2} - x_{02}| \leq h$$

[de sorte que $\forall\, j = 1 \rightarrow 2$: $|\overline{x}_{ij} - \overline{x}_{0j}| \leq h$ c'est-à-dire $\overline{x}_i$ tombe dans $\overline{H}$].

D'un point de vue algorithmique

Les outils utilisés pour le repérage des points suspects sont deux tableaux, entiers bidimensionnels [256] × [256], de référence que nous appelons tr_ADD et tr_SIZ et un tableau, entier unidimensionnel de longueur N, organisateur que nous appelons orga.

Les tableaux tr_ADD, tr_SIZ et orga

Définitions

Le tableau orga est constitué d'une permutation des indices $i = 1 \rightarrow N$ telle que

$$\forall\, i_1, i_2 \in \{1, \cdots, N\} \ , \ i_1 < i_2 :$$

Soit $a =$ orga (i_1) et $b =$ orga (i_2)

Alors ou bien $x_{a1} < x_{b1}$

ou bien $x_{a1} = x_{b1}$ et $x_{a2} \leq x_{b2}$

c'est-à-dire que orga contient tous les "numéros" des points de référence, (une et une seule fois chacun,) ordonnés de façon telle que $\cdots$

$\cdots$ quelque soient i et $j \in [0, 255]$ les numéros des points de référence dont les deux premières coordonnées sont i et j sont rangés à la suite les uns des autres (c'est-à-dire forment un bloc) quelque part dans le tableau.
Tr_ADD (i, j) contient l'"adresse" et tr_SIZ (i, j) la taille de ce bloc, c'est-à-dire :

Le tableau tr-ADD contient en i$^{\text{ème}}$ ligne et j$^{\text{ème}}$ colonne l'indice ("adresse") dans orga du début du bloc formé par les numéros des points dont les deux premières coordonnés sont précisément i et j.

Le tableau tr_SIZ contient en i$^{\text{ème}}$ ligne et j$^{\text{ème}}$colonne la taille du bloc formé dans orga par les numéros des points dont les deux premières coordonnées sont i et j; il s'agit du nombre de points de référence dont les deux premières coordonnées sont i et j.

Construction

La "mise au point" des tableaux débute avec la réalisation, en deux temps, de orga.
Dans un premier temps, orga est initialisé par :

| Pour i allant de 1 à N par pas de 1, faire : orga $[i] = i$

A l'issue de cette section, orga contient dans l'ordre "usuel" arithmétique les numéros des N points de référence : orga $[1] = 1$, orga $[2] = 2$, $\cdots$ et orga $[N] = N$.
Dans un second temps, orga est "arrangé" sur base de la relation no_cmp suivante :

Orga $[p]$ est placé avant orga $[q]$
si et seulement si ou bien Base $(p, 1) <$ Base $(q, 1)$
ou bien Base $(p, 1) =$ Base $(q, 1)$
et Base $(p, 2) \leq$ Base $(q, 2)$

Deux numéros de points p_0 et q_0 tels que Base $(p_0, 1) =$ Base $(q_0, 1)$ et tels que en outre Base $(p_0, 2) =$ Base $(q_0, 2)$ sont placés dans un ordre aléatoire il est vrai, mais aussi, et c'est finalement le plus important puisqu'il s'agit de l'effet recherché, côte à côte au sein d'un même "bloc".

Le tableau tr_SIZ, initialisé à "0 partout", est "rempli", en parcourant l'ensemble des points de référence, considérés un à un par valeurs respectives des deux premières coordonnées val 1 et val 2, avec incrémentation d'une unité de la case (val 1, val 2) de tr_SIZ :

| Pour p allant de 1 à N par pas de 1, faire :
| | val 1 = Base $(p, 1)$
| | val 2 = Base $(p, 2)$
| | tr_SIZ (val 1, val 2) = tz_SIZ (val 1, val 2) + 1

La construction s'achève avec l'élaboration de tr_ADD (sachant tr_SIZ et orga).

offset = 0

Pour i allant de 0 à 255 par pas de 1 :
  Pour j allant de 0 à 255 pas pas de 1 :

    si tr_SIZ $(i,j) = 0$
    alors tr_ADD $(i,j) = 0$
    sinon
      tr_ADD (i,j) = offset
      offset = offset + tr_SIZ (i,j)

Où la variable offset contient une à une au fil de la (double) boucle sur i et j chacune des adresses de début dans orga des blocs associés à (i,j) non vides.

Utilisation

Les tableaux tr_ADD, tr_SIZ et orga fournissent un procédé efficace pour étant conues les valeurs i et j mettre rapidement le doigt sur les points $x. = (i, j, x._3, x._4, \cdots, x._M)$ et rien que sur ceux-ci. Le procédé est défini par :

$$\forall\ (i,j) \in [0,255] \times [0,255] :$$

$$\text{Soit} \quad r_1 = \text{tr\_ADD}\ (i,j)$$

$$r_2 = \text{tr\_SIZ}\ (i,j)$$

$$\text{Alors} \quad \forall\ k = r_1 \rightarrow (r_1 + r_2 - 1) :$$

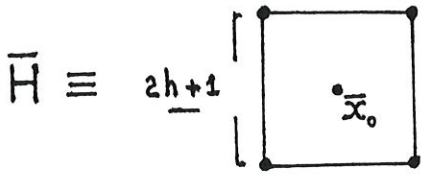
$$\text{Soit } z = \text{orga}\ (k)$$

$$\text{Alors } x_{z1} = i \text{ et } x_{z2} = j.$$

L'étape de recherche est achevée lorsque l'on a déterminé auxquels couples (i,j) il faut appliquer ce procédé pour mettre en évidence l'ensemble des points suspects.

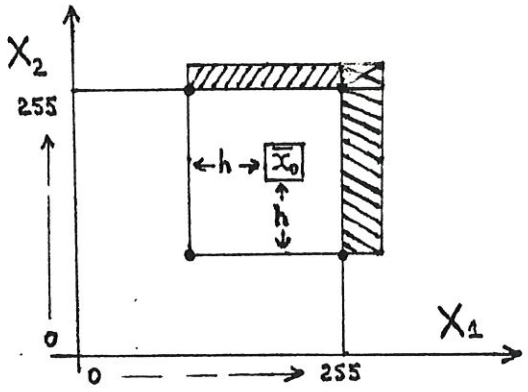
Il nous suffit à ce moment de penser à ce que ces couples (i,j) t.q. $\cdots$ sont définis (en fonction du principe de base de la méthode projective cfr $\cdots$) par les coordonnées des points $\overline{x}_p$ tombant dans la fenêtre de largeur h centrée en $\overline{x}_0$, pour, de manière évidente, porter notre attention sur cette fenêtre $\overline{H}$.

La fenêtre $\overline{H}$ est limitée par les points $\hat{x}_1 = (\overline{x}_{01} - h, \overline{x}_{02} + h)$, $\hat{x}_2 = (\overline{x}_{01} + h, \overline{x}_{02} + h)$, $\hat{x}_3 = (\overline{x}_{01} - h, \overline{x}_{02} - h)$, et $\hat{x}_4 = (\overline{x}_{01} + h, \overline{x}_{02} - h)$,



A bien y réfléchir, cependant, la considération ci-dessus n'est pas tout-à-fait exacte : la fenêtre $\overline{H}$ n'est pas limitée que par ces quatre points de sorte que si elle l'est bien par quatre points, il ne s'agit peut-être pas exactement de ceux-là.
L'espace dans lequel s'inscrit notre modélisation n'est pas l'espace discret à M dimensions tout entier, à savoir $\mathbb{N}^M$, mais une restriction de celui-ci à $[0,255]^M \cdots$
$\cdots$ de sorte que les points $\overline{x}_i$ sont des points non pas de $\mathbb{N}^2$ mais de $[0,255] \times [0,255]$.
La fenêtre $\overline{H}$ doit donc en-outre être, éventuellement, tronquée, pour se situer entièrement dans le plan discret restreint.

Soit $ht = \min(x_{02} + h, 255)$
$gh = \max(0, \ x_{01} - h)$
$bs = \max(0, \ x_{02} - h)$
$dt = \min(x_{01} + h, 255)$



Alors $\overline{H}$ est ainsi définie $\cdots$ par $\cdots$: en haut à gauche $P_1 = (gh, ht)$, en haut à droite $P_2 = (dt, ht)$, en bas à gauche $P_3 = (gh, bs)$ et en bas à droite $P_4 = (dt, bs) \cdots$

$\cdots$ de sorte que : la mise en évidence de l'ensemble des points x_i suspectés de tomber dans H est obtenue par application du procédé (cfr $\cdots$) à chaque couple (i,j) de valeurs dont la considération est amenée par le schéma suivant :

Pour n allant de ht à bs par pas de 1 :
  Pour m allant de gh à dt par pas de 1 :
    considérer $(i,j) = (m,n)$

Sch. 1.

4.3.2 Seconde étape : la distinction parmis ces points "suspects" desquels (ne) tombent effectivement (pas) dans l'hypercube H ?

Formalisation

Les p ($p < q$) points x_i tombant dans H, sont les points d'indice i tels que non seulement $\overline{x}_i \in \overline{M}$ (ils sont suspects) $\cdots$
$\cdots$ mais en outre (les suspicions sont conformes) $\forall\, j = 3 \rightarrow M \;:\; |x_{ij} - x_{0j}| \leq h$.

C'est-à-dire que sur chacun des points suspects est effectué un test par comparaison des $(M-2)$ dernières coordonnées de x_i avec les $(M-2)$ dernières coordonnées de x_0 : de la positivité (resp. négativité) révélée de ce test dépend le comptage (ou non) d'une unité supplémentaire à ρ.

D'un point de vue algorithmique

Soit (i,j) un couple de valeurs mis en évidence lors d'une itération quelconque du schéma (Sch 1), alors tr_ADD (i,j) fournit l'adresse de début et tr_SIZ (i,j) la taille du bloc de orga contenant les numéros des points dont les deux premières coordonnées sont i et j c'est-à-dire que $\cdots$
$\cdots$ la considération de (i,j) amène à suspecter tr_SIZ $(i,j) \equiv z$ points dont les numéros, c'est-à-dire les indices de lignes de la matrice BASE, sont $n^0(1) =$ orga [tr_ADD (i,j)] $n^0(2) =$ orga [tr_ADD $(i,j)+1$] et $\cdots$ $n^0(z) =$ orga [tr_ADD $(i,j)+z-1$].

L'examen particulier de l'un de ces points dont le numéro est, soit, k contribue à incrémenter de une unité un compteur sub lorsque cet examen est OK c'est-à-dire $\cdots$

$$\cdots \text{ ssi } \forall\, \ell = 3 \rightarrow M \;:\; |x_{k\ell} - x_{0\ell}| \leq h\ .$$

De manière évidente, dès qu'il est montré l'existence d'un indice $\ell \in \{3, \cdots, M\}$ tel que $|x_{k\ell} - x_{0\ell}| > h$, l'examen de (le test sur) le point numéro k n'est pas achevé et sub, KO, n'est pas incrémenté.

Le comptage des ρ_{ij} points tombant dans l'hypercube dont la suspection a été amenée par la considération du couple (i,j) est réalisé selon le schéma suivant :

Sch 2.

sub = 0

Pour k allant de 1 à z par pas de 1 :
- Bool = vrai et $\ell = 3$
- tant que (Bool = vrai et $\ell < M$) :
  - si $(x_{k\ell} < x_{0\ell} - h)$ ou $(x_{0\ell} + h < x_{k\ell})$
  - alors Bool = faux
  - $\ell = \ell + 1$
- Si (Bool = vrai)
- alors sub = sub + 1

ρ_{ij} = sub

4.3.3 Intégration des deux étapes

La mesure $\rho(x_0)$ de la densité de la classe considérée autour du point à classer x_0 est finalement réalisée en additionnant les sous-totaux sub compté par (Sch 2) pour chacun des couples (i,j) mis en évidence par (Sch 1).

tot = 0

tant que le schéma (sch 1) met en évidence un couple (i,j)
- compter ρ_{ij} = sub par (Sch 2)
- tot = tot + sub

ρ = tot

* — *

5 La méthode naturelle

5.1 Principe

La méthode naturelle divise l'action en dim étapes où dim est la dimension de l'espace dans lequel s'inscrivent la classe de référence, le point à classer et l'hypercube, c'est-à-dire que dim $= M$ le nombre de caractéristiques explicatives connues de chaque individu.

La j$^{\text{ème}}$ étape ($1 \leq j \leq$ dim) prend en charge $n(j)$ points issus de l'étape $(j-1)$ sur lesquels elle procède à une sélection basée sur le critère : les points ne pouvant appartenir à H du fait de la j$^{\text{ème}}$ coordonnée $\cdots$

$$x_{.j} < x_{0j} - h \quad \text{ou} \quad x_{.j} + h < x_{.j}$$

$\cdots$ ne sont pas pris en compte à l'étape $(j+1)$.

La première étape est appliquée à l'ensemble des points de référence c'est-à-dire $n(1) = N$.

Le nombre de points considérés diminue forcément d'étape en étape, c'est-à-dire que la suite $n(j)$ est décroissante $\cdots$ et converge vers ρ mesure "convoitée" de la densité de la classe E autour de x_0, de sorte que les points qui "restent" au-delà de la dernière étape ($j =$ dim) sont les points qui "tombent" dans l'hypercube.

5.2 Illustration

5.2.1 Achèvement de la méthode en "dim" étapes actives

Soit la base d'entraînement composée de 12 points dont les coordonnées dans l'espace à 7 dimensions, discret restreint, forment la matrice (12×7) suivante :

$$\underline{\text{Base}} \equiv \begin{bmatrix} 32 & 17 & 183 & 25 & 112 & 92 & 48 \\ 56 & 32 & 157 & 42 & 114 & 92 & 55 \\ 42 & 27 & 184 & 36 & 121 & 99 & 52 \\ 38 & 27 & 182 & 37 & 118 & 101 & 53 \\ 39 & 29 & 178 & 38 & 123 & 104 & 52 \\ 37 & 28 & 184 & 35 & 120 & 96 & 54 \\ 52 & 30 & 192 & 40 & 110 & 91 & 49 \\ 60 & 32 & 160 & 44 & 116 & 101 & 51 \\ 40 & 26 & 183 & 38 & 116 & 101 & 51 \\ 36 & 25 & 182 & 38 & 118 & 96 & 48 \\ 41 & 22 & 171 & 28 & 107 & 87 & 36 \\ 36 & 27 & 178 & 32 & 111 & 91 & 40 \end{bmatrix}$$

Alors le comptage des ρ, parmis 12 points tombant dans un hypercube de mesure $h = 3$ centré en le point à classer $\cdots$

$$x_0 = (39, 27, 181, 37, 119, 99, 49)$$

$\cdots$ est réalisé en 7 étapes.

Etape n^0 1

Données. Les premières coordonnées, première colonne de la matrice, et numéros, indices de lignes, associés de chacun des $n(1) = 12$ points de référence.

1	2	3	4	5	6	7	8	9	10	11	12
32	56	42	38	39	37	52	60	40	36	41	36

Traitement. Les points forts de chaque étape j $(1 \leq j \leq \dim)$ sont au nombre de trois.

primo : les données sont triées en ordre croissant des ($j^{\text{ème}}$) coordonnées; lorsque plusieurs coordonnées sont égales, la coordonnée associée au numéro le plus petit est "prise en premier".

1	10	12	6	4	5	9	11	3	7	2	8
32	36	36	37	38	39	40	41	42	52	56	60

deuxio : recherche de la limite inférieure des $j^{\text{ème}}$ coodonnées admissibles, c'est-à-dire de la plus petite des ($j^{\text{ème}}$) coordonnées supérieures à $x_{0j} - h$, identifiée par son numéro d'ordre parmis les données.

$$? \inf(j) \in \{1, \cdots, n(j)\}$$

tertio : recherche de la limite supérieure des ($j^{\text{ème}}$) coordonnées admissibles, c'est-à-dire de la plus grande des ($j^{\text{ème}}$) coordonnées inférieures à $x_{0j} + h$, identifiée par son numéro d'ordre parmis les données.

$$? \sup(j) \in \{1, \cdots, n(j)\}$$

1	10	12	6	4	5	9	4	3	7	2	8
32	36	36	37	38	39	40	41	42	52	56	60

$$\inf(1) = 2 \text{ et } \sup(1) = 9.$$

Résultats

Soit (p, base $(p,1)$) un couple de données situé entre l'ordre (inf (1) =) 2, inclus, et l'ordre (sup (1) =) 9, inclus, alors ...

$$\cdots\ x_{01} - h \leq x_{p1} \leq x_{01} + h$$

$$39 - 3 = 36 \leq x_{p1} \leq 42 = 39 + 3$$

Le point numéro p "appartient" à H du fait de la première coordonnée.

L'étape 1 sélectionne (sup (1) - inf (1) + 1) = 8 points "convenant" $\cdots$
(et rejette les (n(1) - 8) autres) $\cdots$ identifiés par leurs numéros respectifs.

~~36 36 37 38 39 40 41 42~~

10 12 6 4 5 9 11 3

↓

Etape n^0 2

Données. Les secondes coordonnées, échantillon de la seconde colonne de la matrice, et numéros, indices de lignes, associés des points de référence dont les numéros sont "sortis" de l'étape n^0 1, c'est-à-dire $n(2) = 8$.

10 12 6 4 5 9 11 3

25 27 28 27 29 26 22 27

Traitement.

↓

11 10 9 3 4 12 6 5

22 25 26 27 27 27 28 29

$$\begin{array}{cc} 11 & \left[\begin{array}{ccccccc} 10 & 9 & 3 & 4 & 12 & 6 & 5 \\ 25 & 26 & 27 & 27 & 27 & 28 & 29 \end{array}\right] \\ 22 & \end{array}$$

$$x_{02} - h = 24 \ \rightarrow\ \inf(2) = 2\ et\ \sup(2) = 8\ \leftarrow\ 30 = x_{02} + h$$

Résultats. L'étape 2 sélectionne (8 - 2 + 1 =) 7 points dont les numéros sont :

$$10, 9, 3, 4, 12, 6 \text{ et } 5$$

La connaissance de ces numéros autorise la "lecture" dans la matrice BASE des $3^{\text{ème}}$ coordonnées traitées à l'étape 3 : il s'agit de BASE (10, 3), BASE (9,3), BASE (3,3), BASE (4,3), BASE (12,3), BASE (6,3) et BASE (5,3); "Les numéros suivent".

Etape n^0 3

$7 = n(3)$ données :

10	9	3	4	12	6	5
182	183	184	182	178	184	178

$\downarrow$

$$\begin{bmatrix} 178 & 178 & 182 & 182 & 183 & 184 & 184 \\ 5 & 12 & 4 & 10 & 9 & 3 & 6 \end{bmatrix}$$

$7 = n(4)$ résultats : 5 12 4 10 9 3 6

Etape n^0 4

En règle générale, on travaille de moins en moins, d'étape en étape. Il arrive cependant qu'aucun point ne soit écarté au cours d'une étape : c'est le cas à l'étape 3 où $x_{03} - h \leq 178$ et $184 \leq x_{03} + h$ puisque $x_{03} = 181$ et $h = 3$ de sorte que $n(4) = n(3) = 7$.

5	12	4	10	9	3	6
38	32	37	38	38	36	35

$\downarrow$

$$\begin{array}{c} 32 \\ 12 \end{array} \begin{bmatrix} 35 & 36 & 37 & 38 & 38 & 38 \\ 6 & 3 & 4 & 5 & 9 & 10 \end{bmatrix}$$

$\rightarrow$ 6 3 4 5 9 10

Etape n^0 5

6	3	4	5	9	10
120	121	118	123	116	118

↓

116	118	118	120	121	123
9	4	10	6	3	5

Etape n^0 6

9	4	10	6	3

101	101	96	96	99

↓

96	96	99	101	101
6	10	3	4	9

Etape n^0 7

↓

6	10	3	4	9
54	48	52	53	51

↓

48	51	52	53	54
10	9	3	4	6

Les points qui restent au-delà de la 7$^{\text{ème}}$ et dernière étape, à savoir les points numéros 3, 9 et 10 sont au nombre de 3. Il s'agit des points qui tombent dans l'hypercube.

$$\rho = \sup(7) - \inf(7) + 1 = 3 - 1 + 1 = 3 .$$

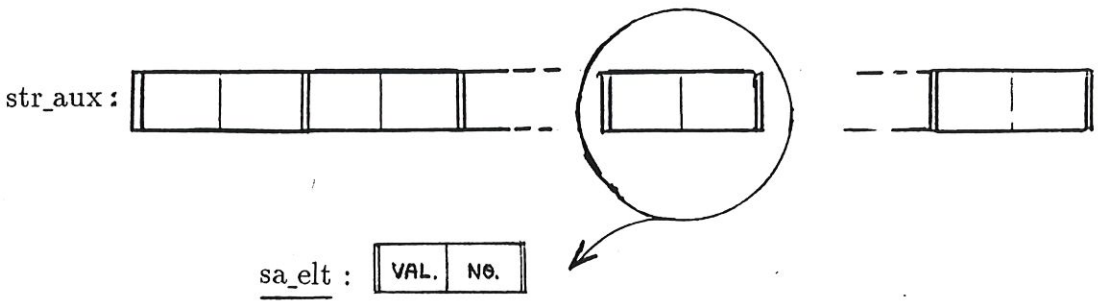
5.3 Algorithme

La méthode utilise principalement un outil mis au point grâce à une utilisation appropriée des "concepts" de la programmation moderne, en particulier pour notre propos du langage C.

5.3.1 La structure str_aux

Il s'agit de construire une structure apte, de la meilleure façon possible, à contenir les données de chaque étape, à en permettre un traitement efficace, précis et rapide, et à mettre en évidence les résultats en facilitant la transition entre deux étapes consécutives.

Soit un tableau dont les éléments sont des "couples" (records) définis de manière à pouvoir contenir d'une part une valeur x_{ij} de la matrice BASE, c'est-à-dire une coordonnée d'un point, et d'autre part un indice de ligne, c'est-à-dire un numéro d'un point : nous appelons str_aux ce tableau.



où $\longrightarrow$ val est une coordonnée d'un point, c'est-à-dire que val peut prendre chaque valeur comprise entre 0 et 255,
et $\longrightarrow$ no est un numéro d'un point c'est-à-dire $1 \leq$ no $\leq N$, le nombre de points de la base d'entraînement; il n'y a aucune limite supérieure sur N de sorte que $\cdots$

$\cdots$ nous définissons le type, sa_elt, d'un élément de str_aux comme un record composé d'une part d'une variable val de type unsigned char et d'autre part d'une variable no de type unsigned int.

Les "notations" str_aux $[p] \rightarrow$ val et str_aux $[p] \rightarrow$ no identifient les parties val et, respectivement, no du $p^{\text{ème}}$ élément du tableau.

Alors str_aux autorise un accomplissement efficace de chacune des dim étapes actives de la méthode.

A chaque étape j ($1 \leq j \leq$ dim) str_aux est défini de longueur $n(j)$ notée par, soit, n, pour alléger la notation, et initialisé par la section :

```
| pour p allant de 1 à n par pas de 1
| faire :
|     | str_aux [p] → val = Base (i_p, j)
|     | str_aux [p] → no = i_p
```

où $i_1, i_2, \cdots$ et i_n sont les numéros des points considérés à l'étape j.
En particulier à l'étape 1 il s'agit des indices i de 1 à N, sans exception.

5.3.2 Opérations sur str_aux

Les principales opérations qu'il est nécessaire de pouvoir effectuer sur str_aux pour que chaque étape j puisse être menée à son terme sont :

primo : une opération de tri par ordre croissant

secundo : une opération de recherche d'une valeur particulière parmis les valeurs (str_aux $[p] \rightarrow$ val) $p : 1 \rightarrow n$.

Tri rapide

L'opération de tri est concrétisée par un appel à la fonction QSORT prédéfinie du C : QSORT est une implémentation de l'algorithme de tri rapide "quick sort". Elle trie les éléments d'un tableau par des appels successifs à une fonction de comparaison définie par l'utilisateur, soit elt_cmp pour le cas de str_aux.

La fonction elt_cmp compare deux éléments str_aux $[p]$ et str_aux $[q]$ du tableau str_aux et retourne un entier < 0 ou un entier > 0 selon que l'élément p doive apparaître avant l'élément q dans le tableau trié ou que l'élément q doive apparaître avant l'élément p.

Elt_cmp est définie par :

```
si (str_aux [p] → val = str_aux [q] → val)
alors retourner (str_aux [p] → no - str_aux [q] → no)
sinon retourner (str_aux [p] → val - str_aux [q] → val)
```

Recherche dichotomique

L'opération de recherche d'une valeur particulière ··· est accomplie par un appel à la fonction binary search *b_s* décrite par :

B_s utilise un algorithme de recherche dichotomique pour effectuer une recherche dans le tableau str_aux et retourner la position dans str_aux du premier élément rencontré dont la partie val correspond à la clé de recherche. Les éléments du tableau doivent être triés par ordre croissant avant l'appel de *b_s*. Si aucun élément ne convient *b_s* retourne (-1).

```
low = 1
high = n
tant que (low ≤ high)
faire
    mid = (low + high) / 2
    si clé < (str_aux [mid] → val)
    alors high = mid - 1
    sinon si clé > (str_aux [mid] → val)
        alors low = mid + 1
        sinon retourner (mid)
retourner (-1)
```

L'appel *b_s* (clé, str_aux, *n*) effectue la recherche ···

$$\cdots ?\ p \in \{1, \cdots, n\} \text{ tel que str\_aux } [p] \rightarrow \text{ val} = \text{clé}$$

5.3.3 Utilisation pour l'accomplissement d'une étape

Les opérations de tri rapide QSORT et de recherche dichotomique *b_s* sur str_aux sont exploitées dans l'élaboration des phases principales de l'action réalisée au cours de chacune des étapes.

QSORT

Soit str_aux initialisé à l'étape k :

str_aux :	$B(i_1,k)$	i_1	$B(i_2,k)$	i_2				$B(i_n,k)$	i_n

où $i_1, i_2, \cdots$ et i_n sont les numéros des points considérés à l'étape k.

Illustration : str_aux :

93	12	52	14	112	9	128	7	18	21	212	13	93	5	112	17	105	19

Alors Qsort ↓

str_aux :	$B(j_1,k)$	j_1	$B(j_2,k)$	j_2				$B(j_n,k)$	j_n

où $\{j_1, j_2, \cdots, j_n\}$ est une permutation de $\{i_1, i_2, \cdots, i_n\}$ telle que

Soit $p, q \in \{1, 2, \cdots, n\}$ tel que $p < q$
Alors ou bien $B(j_p, k) < B(j_q, k)$
ou bien $B(j_p, k) = B(j_q, k)$ et $j_p < j_q$

de sorte que : str_aux trié en ordre croissant :

18	21	52	14	93	5	93	12	105	19	112	9	112	17	128	7	212	13

Ai_excess

La procédure ai_excess effectue la recherche parmis $\{p : 1 \rightarrow n\}$, sur str_aux trié en ordre croissant de l'indice p_0 (la place dans str_aux) d'un élément sa_elt dont la partie val est une plus proche par excès d'une clé de recherche, c'est-à-dire "une" plus directement supérieure ou égale à "clé", minimal, c'est-à-dire tel que si plusieurs éléments val (p) sont identiquement proche de "clé" alors ai_excess retient l'indice le plus bas.

$$? \; p_0 = \min \{p \text{ tel que str\_aux } [p] \rightarrow \text{ val } = \overline{\text{clé}} \}$$

$$\text{où } \overline{\text{clé}} \; = \min\{B(j., k) \geq \text{ clé } \}$$

La procédure ai_excess est implémentée sur le schéma suivant :

```
| écart = 0
|
| répéter
|     | p = b_s (clé + écart, str_aux, n)
|     | écart = écart + 1
| tant que b_s retourne p égal à (-1)
```

A ce stade b_s a finalement retourné p tel que str_aux $[p] \rightarrow$ val $= \overline{\text{clé}}$
où $\overline{\text{clé}}$ est égal à l'argument (clé + écart) de ce dernier appel à b_s.
Il reste à passer à l'indice minimal p_0.

```
| tant que (str_aux [p − 1] → val = str_aux [p] → val)
|     | p = p − 1
|     | sauf si p = 0
|     | dans lequel cas : retourner (0)
|
| retourner (p)
```

Ai_default

La procédure ai_default effectue la recherche, parmis $\{p : 1 \rightarrow n\}$, sur str_aux trié en ordre croissant de l'indice p_0 (la place dans str_aux) d'un élément sa_elt dont la partie val est "une" plus proche par défaut d'une clé de recherche, c'est-à-dire "une" plus directement inférieure ou égale à "clé", maximal, c'est-à-dire tel que si plusieurs éléments val (p) sont identiquement proche de "clé" alors ai_default retient l'indice le plus haut.

$$? \; p_0 = \max\{p \text{ tel que str\_aux } [p] \rightarrow \text{ val} = \underline{\text{clé}}\}$$

$$\text{où } \underline{\text{clé}} \; = \max\{B(j., k) \leq \text{ clé }\}$$

La procédure ai_default est implémentée sur le schéma suivant :

```
| écart = 0
|
| répéter
|     | p = b_s (clé - écart, str_aux, n)
|     | écart = écart + 1
| tant que b_s retourne p égal à (-1)
```

A ce stade b_s a finalement retourné p tel que str_aux $[p] \rightarrow$ val = $\underline{\text{clé}}$ où $\underline{\text{clé}}$ est égal à l'argument (clé – écart) de ce dernier appel à b_s.
Il reste à passer à l'indice maximal p_0.

```
| tant que (str_aux [p + 1] → val = str_aux [p] → val)
|     | p = p + 1
|     | sauf si p = n .
|     | dans lequel cas : retourner (n)
|
| retourner (p)
```

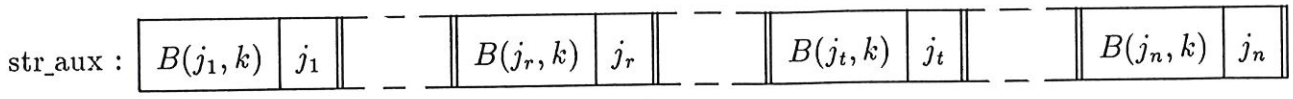
Utilisation

Les appels successifs à ai_excess et ai_default suivant $\cdots$

$$r = \inf(k) = \text{ai\_excess (cle\_l, str\_aux, } n) \text{ où cle\_l} = \max(0,\ x_{0k} - h)$$

$$t = \sup(k) = \text{ai\_default (cle\_h, str\_aux, } n) \text{ où cle\_h} = \min(x_{0k} + h, 222)$$

$\cdots$ définissent, parmis $\{p : 1 \rightarrow n\}$, les limites inférieure et, respectivement, supérieure des ($k^{\text{ème}}$) coordonnées admissibles de sorte que :



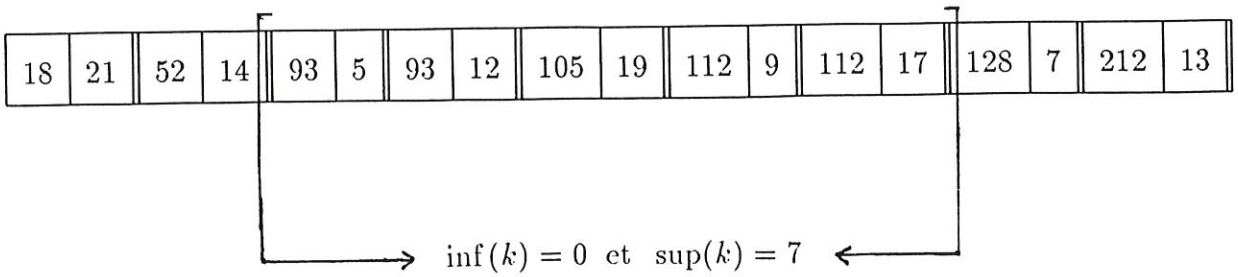
$$\forall\ s \in \{r, r+1, \cdots, t-1, t\} : x_{0k} - h \leq B(j_s, k) \leq x_{0k} + h$$

c'est-à-dire les points numéro $j_r, j_{r+1}, \cdots, j_{t-1}$ et j_t "appartiennent" à H du fait de la $k^{\text{ème}}$ coordonnée.

Illustration

18	21	52	14	93	5	93	12	105	19	112	9	112	17	128	7	212	13

Soit $x_{0k} = 102$ et $h = 12$ Alors $x_{0k} - h = 90$ et $x_{0k} + h = 114$ de sorte que ai_excess (90, str_aux, 9) et ai_default (114, str_aux, 9) amènent :



L'étape k sélectionne 7 - 3 + 1 = 5 points dont les numéros, indices de ligne dans la matrice BASE, soit 5, 12, 19, 9 et 17.

Clôture

1 Le programme CES

Le programme CES dont la source CES.C est donné en annexe, se propose d'appliquer l'une (au choix de l'utilisateur) des méthodes projective et naturelle pour mesurer la densité d'une classe autour d'un individu, c'est-à-dire pour compter le nombre de points parmi ceux d'une base d'entraînement qui tombent dans un hypercube de mesure h, connue, centré en un point supplémentaire.

1.1 Lecture des données

La fonction dat_lec assimile les données propres à une séquence CES, introduites au clavier par l'utilisateur au fur et à mesure de demandes explicites portées à l'écran par le programme, selon un schéma précis duquel résulte le "dialogue" programme_utilisateur suivant :

Prog. > Dimension de l'espace des coordonnées $\cdots$?
Util. > "M" $\hookleftarrow$

Prog. > Nom du fichier_base contenant le training_set ?
Util. > "$\rightsquigarrow$ fic_base.dat" $\hookleftarrow$
où $\rightsquigarrow$ constitue le chemin d'accès au fichier fic_base.dat

Prog. > Centre (point à classer) de l'hypercube ?
Util. > "x_{01}" $\hookleftarrow$ "x_{02}" $\hookleftarrow \ldots$ "x_{0M}" $\hookleftarrow$

Prog. > Mesure (H) de l'hypercube ?
Util. > "h" $\hookleftarrow$

Il faut remarquer que le programme ne s'enquère pas du nombre N de points du training-set; il ne s'en passe pas pour autant : celui-ci est compté à la lecture du fichier fic_base.dat lors du remplissage, ligne par ligne, de la matrice BASE.
L'utilisateur n'est donc pas obligé de connaître le nombre exact de points de la base d'entraînement; ceci est fort "assouplissant", surtout lorsque dans d'importantes applications les classes de référence sont de grandes tailles.

Le prix à payer pour cette "commodité" est que la fin du fichier fic_base.dat soit atteinte (en lecture, il suffit de tester si la valeur transmise par le caractère reçu est égale à la valeur prédéclarée EOF) exactement après la dernière coordonnée du dernier point ···

1.2 Traitement

La dernière question posée par le programme, au terme de la procédure d'acquisition des données, concerne le choix de la méthode ···

Prog. > Method ?

··· à laquelle l'utilisateur répond par 0 ou 1 suivant qu'il veuille générer un traitement des informations par la méthode projective ou naturelle.

[type 1 (0) and return for Nat. (Proj.)] **? 0/1**

1.3 Ecriture des résultats

La méthode choisie est appliquée et si le programme s'est déroulé normalement les résultats sont portés à l'écran. Il s'agit primo de la description des modalités d'exploitation de la séquence CES clôturée, objet du dialogue préliminaire programme_utilisateur, deuxio de la valeur prise par l'estimateur de densité (résultat principal = nombre de points tombant dans l'hypercube) et tertio d'indications sur le temps nécessité par l'exécution du comptage.

Au terme de l'inscription à l'écran des résultats l'utilisateur a la possibilité de laisser une trace de ceux-ci dans un fichier de son choix.

Prog. > Trace ?

Il peut accepter ou refuser cette invitation. [type 1 (0) and return for Yes (No)]

S'il refuse, le programme est terminé.

Si, au contraire, il accepte, le dialogue programme_utilisateur reprend, duquel l'objet est à ce moment le nom du fichier choisi par l'utilisateur pour laisser la trace de la séquence.

Prog. > Nom du fichier trace ?
Util. > " $\rightsquigarrow$ fic_trace.res" $\hookleftarrow$
où $\rightsquigarrow$ constitue le chemin d'accès au fichier fic_trace.res.

Les résultats sont portés à la fin de fic_trace.res.
$\cdots$ (si le fichier n'existe pas il est créé.)
La séquence du programme CES est terminée.

La description ci-dessus correspond au déroulement normal du programme $\cdots$

1.4 Interruptions

Les erreurs gérées par le programme sont de l'ordre de la gestion, d'une part, de la mémoire et, d'autre part, des fichiers : une interruption est provoquée (avec envoi d'un message $\cdots$) lorsque l'un des fichiers fic_base.dat et fic_trace.res est inaccessible ou lorsque il n'est pas possible d'allouer suffisamment de mémoire dynamique où "placer" BASE, str_aux, tr_SIZ, tr_ADD ou orga : ces dernières peuvent cependant être évitées au maximum par une compilation du programme source en mode "Huge", lequel autorise, au détriment de la place accordée, en mémoire vive, au code, des tailles de tableaux dépassant 64 Ko, pour les très grosses applications.

Les erreurs issues d'une utilisation non rationnelle de la séquence CES (c'est-à-dire non en concordance avec le modèle théorique développé) ne sont pas gérées par le programme; il ne nous était pas possible de prédéfinir une interruption associée à chacune des erreurs possibles à ce niveau, trop nombreuses.

* — *

2 Comparaison des méthodes

Le programme CES permet de tester et de comparer le comportement de chacune des méthodes naturelle et projective vis-à-vis de variations des modalités d'exploitation de la séquence CES. Pour notre propos, nous avons principalement étudié la relation entre le temps d'exécution et la mesure h de l'hypercube, toutes "choses" par ailleurs étant égales. Les conclusions de cette étude sont théoriquement "prévisibles", c'est-à-dire essentiellement "explicables" parce que l'histoire de la pratique scientifique montre qu'un évévement "extra-ordinaire" est plus souvent constaté et expliqué (après $\cdots$) que prévu (avant $\cdots$).

Le temps nécessaire à l'exécution d'une séquence CES par la méthode projective croît de manière quasi quadratique avec la mesure h de l'hypercube.
Lorsque h est de faible valeur la fenêtre $\overline{H}$ est "petite" par rapport au plan principal restreint [0,255] $\times$ [0,255] de sorte que les cases (i,j), des tableaux tr_ADD et tr_SIZ, visitées lors de l'étape n^0 1 de repérage des points "suspects" sont peu nombreuses, soit au nombre de V_A dans tr_ADD et de V_S dans tr_SIZ avec $V_A \leq V_S$.

Dès que h augmente le taux de cases (i,j) visitées par le programme dans le tableau tr_SIZ, parmis le nombre total 256^2, croît en fonction de la règle suivante :

$$\begin{aligned} \text{Si } h \rightsquigarrow h+1 \text{ alors } V_S = (2h+1)^2 \rightsquigarrow V_S &= (2(h+1)+1)^2 \\ &= (2h+3)^2 \\ &= \underline{\underline{4h^2+12h+9}} \end{aligned}$$

Les tableaux tr_ADD et tr_SIZ sont en outre définis de manière telle que en pratique un grand nombre de cases contiennent une valeur exactement nulle; de fait tr_ADD (i,j) et tr_SIZ (i,j) sont "égal à 0" dès que aucun point de la base de référence n'a pour deux premières coordonnées le couple (i,j).

Parmis les V_S cases visitées de tr_SIZ, un certain pourcentage (voire un pourcentage certain) le sont donc "pour du beurre" : le chronomètre, cependant, ne fait pas la distinction - il n'a d'ailleurs pas à la faire - et le temps nécessaire à la réalisation de ces opérations "inutiles" est - comme il se doit - "compté avec $\cdots$".

L'utilisation, dans le cas de h "grand", de la méthode naturelle permet d'éviter la croissance abusive du temps d'exécution d'une séquence CES; il n'y a, par la méthode naturelle, aucun travail "pour rien" mais il reste une ombre au tableau de l'efficacité de cette alternative $\cdots$: le prix à payer est une limitation du nombre nbc de points à classer.

Il faut pour établir cette restriction constituer et admettre le fait selon lequel les méthodes naturelle et projective sont de types différents non seulement de manière conceptuelle mais surtout par leurs natures respectives.

La méthode projective proj_meth est une méthode "à préprocessing" c'est-à-dire qu'un certain nombre d'outils (tr_SIZ, tr_ADD et orga) sont construits avant toute autre chose et pour la durée entière du processus de classement : la même boîte à outil est utilisée par chaque séquence CES sur chacun des points à classer.

La méthode naturelle nat_meth est une méthode "sans préprocessing" c'est-à-dire que les outils (str_aux $\cdots$) utilisés par une séquence CES sont construits "au fur et à mesure" et pour la seule durée de cette séquence isolée.

Soit T le temps de réalisation du préprocessing de proj_meth
t le temps d'une séquence CES par proj_meth
t_1 et t_2 les temps de construction $\cdots$ et d'utilisation $\cdots$ pour nat_meth.
Alors le temps nécessaire à la mesure de la densité d'une classe autour de 1 individu, par chacune des méthodes projective ou naturelle, est donné, respectivement, par :

$$(\text{proj\_meth } >)\ T + t \quad \text{et} \quad t_1 + t_2\ (< \text{ nat\_meth}).$$

Les temps généraux nécessaires à la mesure de nbc densités associées à une classe de référence et à, respectivement, chacun de nbc individus supplémentaires sont donnés par :

$$(\text{proj\_meth. } >)\ \ T + nbc.t$$

$$(\text{nat\_meth. } >)\ \ nbc.(t_1 + t_2) = nbc.t_1 + nbc.t_2$$

La préférence d'un utilisateur d'un processus de classement se portera sur la méthode naturelle plutôt que sur la méthode projective jusqu'à un certain nombre n de points à classer tel que $\cdots$

$$T + n.t = n.t_1 + n.t_2$$

$$\longrightarrow \quad T = n.(t_1 + t_2 - t)$$

$$n = \frac{T}{t_1 + t_2 - t}$$

où T, t_1 et t_2 sont, toutes proportions gardées, indépendants de la mesure h de l'hypercube, alors que $\cdots$

$\cdots t$ croît avec h de sorte que $\cdots$

$$\text{Si } h \uparrow \text{ alors } n = \frac{T}{t_1 + t_2 - (\uparrow)}$$

$$n = T/(\downarrow) \text{ c'est-à-dire } n \uparrow .$$

* — *

3 Conclusions et perspectives

Il ressort de la comparaison que la méthode naturelle est préférable à la méthode projective jusqu'à un seuil critique dont la mesure (exprimée en un nombre de points à classer) "recule" avec toute augmentation de la valeur h.

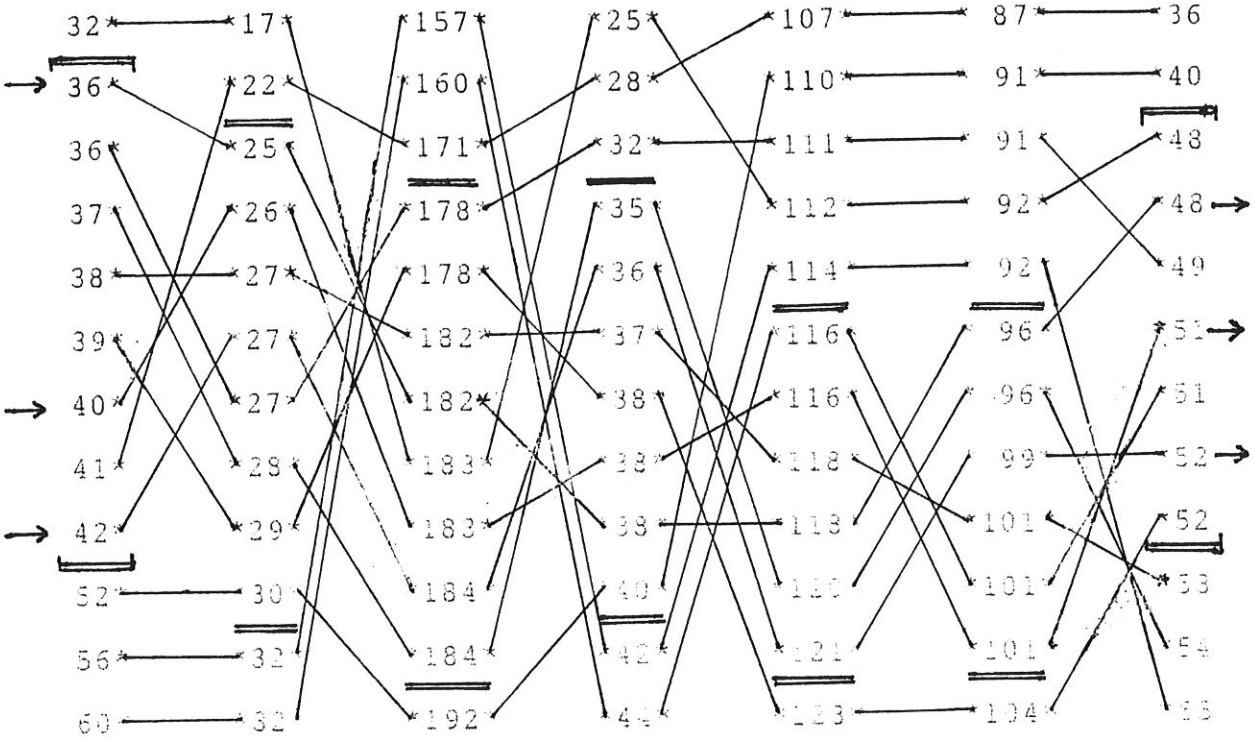
Une méthode meilleure dans tous les cas existe sans doute quelque part à la frontière des natures ou principes respectifs de chacune des méthodes projective et naturelle : c'est-à-dire que si l'on peut concevoir un algorithme "à préprocessing" d'une part (construction d'une boîte à outils utilisée nbc fois, de façon à éviter la limitation exercée sur nat_meth.) et d'autre part tel que l'utilisation de la boîte à outils soit indépendante de la mesure h de l'hypercube (il faut impérativement éviter que, toutes "choses" par ailleurs étant égales, une augmentation de h n'oblige à un travail supplémentaire même léger), alors la méthode associée à cet algorithme doit être "optimale".

Une direction de recherche dans laquelle nous ne saurions manquer d'espérer ...
... un engagement prochain est constituée par l'illustration ($N = 12$ et $M = 7$) ci-dessous.

Soit la base d'entraînement composée de 12 points dont les coordonnées dans l'espace à 7 dimensions, discret restreint, formant la matrice (12×7) suivante :

$$\text{Base} \equiv \begin{bmatrix} 32 & 17 & 183 & 25 & 112 & 92 & 48 \\ 56 & 32 & 157 & 42 & 114 & 92 & 55 \\ 42 & 27 & 184 & 36 & 121 & 99 & 52 \\ 38 & 27 & 182 & 37 & 118 & 101 & 53 \\ 39 & 29 & 178 & 38 & 123 & 104 & 52 \\ 37 & 28 & 184 & 35 & 120 & 96 & 54 \\ 52 & 30 & 192 & 40 & 110 & 91 & 49 \\ 60 & 32 & 160 & 44 & 116 & 101 & 51 \\ 40 & 26 & 183 & 38 & 116 & 101 & 51 \\ 36 & 25 & 182 & 38 & 118 & 96 & 48 \\ 41 & 22 & 171 & 28 & 107 & 87 & 36 \\ 36 & 27 & 178 & 32 & 111 & 91 & 40 \end{bmatrix}$$

Alors un tri de chacune des colonnes de la matrice avec maintient du lien entre les 7 coordonnées de chacun des 12 points génère le réseau ···



Le point à classer ···

$$x_0 = (39, 27, 181, 37, 119, 99, 49)$$

··· et la mesure $h(= 3)$ de l'hypercube...

... définissent sur ce réseau 7 portes $[\![\cdots]\!]$.

Les points de la base d'entraînement dont le lien ···→··· entre les coordonnées passe au travers de chacune de ces portes sont les points qui "tombent" dans l'hypercube ···

* — * — *

BIBLIOGRAPHIE

Références propres au domaine informatique

J. ARSAC, "La programmation", Cedic.

B. MEYER and C. BAUDOIN, Electricité de France.
"Méthodes de Programmation".
Eyrolles : Collection de la Direction des Etudes et Recherches d'E.D.F..

A.B. TUCKER, Colgate University.
"Programming languages".
Mc Graw Hill.
- "Fortran, Une introduction"."Mc Graw Hill : Les langages de programmation".
- "Langage C, Une introduction". "Mc Graw Hill : Les langages de programmation".

F. FICHEUX-VAPNE, Electricité de France.
"Fortran 77, Guide pour l'écriture de programmes portables".
Eyrolles : Collection de la Direction des Etudes et Recherches d'E.D.F..

B.W. KERNIGHAN and D.M. RITCHIE.
"Le langage C".
Manuels Informatiques Masson.

C.L. TONDO and S.E. GIMPEL.
"Le langage C, solutions (des exercices proposés par Kernighan et Ritchie)".
Manuels Informatiques Masson.

J.-J. MEYER.
Initiation à Turbo C (toute la programmation en langage C).
Editions Radio.

Documentation/Turbo C 2.0 by BORLAND.

Références propres au domaine statistique

A. MONTFORT,
"Cours de statistique" (E.N.S.A.E.),

M. VOLLE,
"Analyse des données" (E.N.S.A.E.),
Economica. Collection "Economie et statistiques avancées".

A. HARDY et J.-P. RASSON,
"Une nouvelle approche des problèmes de classification automatique".
Statistique et analyse des données, 7, 1982.

A. HARDY,
"Statistique et classification automatique : ···
··· un modèle, un nouveau critère, des algorithmes et des applications".
PhD Thesis, Facultés Universitaires de Namur, Belgique, 1983.

P. BAUFAYS et J.-P. RASSON,
"Une nouvelle règle de classement utilisant l'enveloppe convexe et la mesure de Lebesgue".
Statistique et analyse des données, 9, 1984.

P. BAUFAYS,
"Une nouvelle règle géométrique en analyse discriminante : ···
··· critère de classement, affectation, courbes de décision et applications réelles".
PhD Thesis, Facultés Universitaires de Namur, Belgique, 1985.

B.W. SILVERMAN, School of mathematics, University of Bath, U.K..
"Density Estimation for Statistics and Data Analysis".
Chapman and Hall.

J.-P. RASSON,
"Analyse discriminante pour le Processus de Poisson non homogène".
Rapport interne, Facultés Universitaires de Namur, Belgique, 1992.

S. OCHAO,
"Une nouvelle méthode de "classification" ···
··· basée sur le processus de Poisson non homogène".
Mémoire, Facultés Universitaires de Namur, Belgique, 1992.

Annexe

CES.C

```
/* BEGIN CES.C */


#include <stdio.h>
#include <stdlib.h>
#include <time.h>


typedef struct
  { unsigned char val ;
    unsigned int   no ;
  } sa_elt ;


void          pres_prog  ( ) ;

void          dat_lec    ( ) ;
void          nat_meth   ( ) ;
void          proj_meth  ( ) ;
void          ecr_res    ( ) ;

void          clot_prog  ( ) ;

int           no_cmp     ( unsigned int * , unsigned int * ) ;

int           elt_cmp    ( sa_elt * , sa_elt * ) ;

unsigned int ai_excess  ( unsigned char , sa_elt * , unsigned int ) ;
unsigned int ai_default ( unsigned char , sa_elt * , unsigned int ) ;

int           b_s        ( unsigned char , sa_elt * , unsigned int ) ;


unsigned char dim ;

char          fb_nom[80] ;
unsigned char **base ;

unsigned char *c_cube ;
unsigned char m_cube ;

unsigned int  nbr , res ;

clock_t       beg , end ;
clock_t       P_b , P_e ;

unsigned int  remake ;
unsigned int  stop = 1000 ;

unsigned char altern = 1 ;
```

```
/* begin main */

main ( )

  {
   clrscr () ;
   pres_prog () ;
   puts (" Let's go ! ") ;

   dat_lec ( ) ;

   if (altern)
      nat_meth ( ) ;
   else
      proj_meth ( ) ;

   ecr_res ( ) ;

   puts (" That' s all ! ") ;
   clot_prog () ;
  }

/* end main */
```

```
/* begin pres_prog */

void pres_prog ( )

  {
   printf ("\n\t     Outil pour la resolution ...") ;
   printf ("\n\t ... d' un probleme de classement ...") ;
   printf ("\n\t   ... en Analyse Discriminante ! \n") ;

   printf ("\n\t Mesure de la densite d' une classe ...") ;
   printf ("\n\t   ... autour d' un individu ! \n") ;

   printf ("\n\t      Calcul du nombre de points ...") ;
   printf ("\n\t  ... parmis N d' une base d' entrainement ...") ;
   printf ("\n\t      ... tombant dans un hypercube ...") ;
   printf ("\n\t  ... de mesure h centre en le point a classer ! \n") ;

   printf ("\n\n\n CES by proj._ OR nat._ method . \n\n") ;

   puts (" Press any key for continue ! ") ;
   getch () ;
   clrscr () ;
  }

/* end pres_prog */
```

```
/* begin dat_lec */

void dat_lec ( )

  {
   FILE          *f_base ;
   unsigned int i_1 , i_2 ;

   printf ("\n\t >>> \n") ;

   printf ("\nDimension de l' espace des coordonees ... ? \n") ;
   scanf ("%u" , &dim ) ;

   printf ("\nNom du fichier_base contenant le training_set ? \n") ;
   scanf ("%s" , &fb_nom ) ;

   c_cube = (unsigned char *) malloc (dim * sizeof(unsigned char)) ;

   printf ("\nCentre (Point a classer) de l' hypercube ? \n") ;
   for (i_1 = 0; i_1 < dim; i_1++)
       scanf ("%u" , c_cube+i_1 ) ;

   printf ("\nMesure (H) de l' hypercube ? \n") ;
   scanf ("%u" , &m_cube ) ;

   printf ("\nMethod ? [ type 1 (0) and return for Nat. (Proj.) ] \n") ;
   scanf ("%u" , &altern ) ;

   clrscr () ;

   f_base = fopen (fb_nom , "r+t") ;

   if ( f_base == NULL )
     {
      puts (" fopen ERROR : unknown or ... file . ") ;
      puts (" Program aborted ! ") ;
      exit (1) ;
     }

   base = (unsigned char **) malloc (1 * sizeof(unsigned char *)) ;

   for (i_2=0 ; !(feof(f_base)) ; i_2++)
      {
       unsigned int i_3 ;

       base = realloc ( base , ((i_2 + 1) * sizeof(unsigned char *)) ) ;

       base[i_2] = (unsigned char *) malloc (dim * sizeof(unsigned char))

       if ( base [i_2] == NULL )
         {
          puts (" base alloc ERROR : unsufficient memory . ") ;
          puts (" Program aborted ! ") ;
          exit (1) ;
         }

       for (i_3=0 ; i_3<dim ; i_3++)
           fscanf ( f_base , "%u" ,   (*(base+i_2))+i_3 )  ;
      }

    fclose ( f_base ) ;

    nbr = i_2 ;
   }

  /* end dat_lec */
```

```
/* begin nat_meth */

void nat_meth ( )

  {
   sa_elt         *str_aux ;
   unsigned int   j_1 ;

   unsigned char cle_l , cle_h ;
   unsigned int  low , high ;

   unsigned int  J ;

   unsigned int  nbres ;

beg = clock () ;

for ( remake=0 ; remake<stop ; remake++)
  {
   nbres = nbr ;

/* J = 1 ; > */

/* begin step_1 */

   str_aux = (sa_elt *) malloc (nbres * sizeof(sa_elt)) ;

   if ( str_aux == NULL )
     {
      puts (" str_aux alloc ERROR : unsufficient memory . ") ;
      puts (" Program aborted ! ") ;
      exit (1) ;
     }

   for (j_1=0 ; j_1<nbres ; j_1++)
      {
       (str_aux + j_1) -> val = *(*(base + j_1)) ;
       (str_aux + j_1) -> no = j_1 ;
      }

   qsort ( str_aux , nbres , sizeof(sa_elt) , elt_cmp ) ;

   cle_l = max ( 0 , ((*c_cube) - m_cube) ) ;
   cle_h = min ( ((*c_cube) + m_cube) , 255 ) ;

   low = ai_excess ( cle_l , str_aux , nbres ) ;
   high = ai_default ( cle_h , str_aux , nbres ) ;

   nbres = (high - low) + 1 ;

 /* end step_1 */
```

```
/* J : 2 -> dim ; > */

   for (J=2 ; J<=dim ; J++)

/* begin step_J */

       {
        unsigned int *copy ;

        unsigned int j_2 , j_3 ;

        copy = (unsigned int *) malloc (nbres * sizeof(unsigned int)) ;

        for (j_2=0 ; j_2<nbres ; j_2++)
           {
            unsigned int ind_1 ;

            ind_1 = low + j_2 ;
            *(copy + j_2) = (str_aux + ind_1) -> no ;
           }

        str_aux = realloc ( str_aux , (nbres * sizeof(sa_elt)) ) ;

        for (j_3=0 ; j_3<nbres ; j_3++)
           {
            unsigned int ind_2 ;

            ind_2 = *(copy+j_3) ;

            (str_aux + j_3) -> val = *((*(base+ind_2))+(J-1)) ;
            (str_aux + j_3) -> no = ind_2 ;
           }

        free (copy) ;

        qsort ( str_aux , nbres , sizeof(sa_elt) , elt_cmp ) ;

        cle_l = max ( 0 , ((*(c_cube+J-1)) - m_cube) ) ;
        cle_h = min ( ((*(c_cube+J-1)) + m_cube) , 255 ) ;

        low = ai_excess ( cle_l , str_aux , nbres ) ;
        high = ai_default ( cle_h , str_aux , nbres ) ;

        nbres = (high - low) + 1 ;
       }

 /* end step_J */

    free (str_aux) ;

  }

 end = clock () ;

    res = nbres ;
  }

 /* end nat_meth */
```

```
/* begin proj_meth */

void proj_meth ( )

  {
   /* all proj_meth */

   unsigned int  **tr_ADD ;
   unsigned char **tr_SIZ ;
   unsigned int  *orga ;

   /* pre_pro */

   unsigned int  offset ;
   unsigned int  j_1 , j_2 , j_3 , j_4 , j_5 ;

   /* meth */

   unsigned char cc_1 , cc_2 ;
   unsigned char ht , bs , gh , dt ;
   unsigned int  I , J ;
   unsigned int  tot ;
```

```
/* begin pre_pro */
   P_b = clock () ;

   tr_SIZ = (unsigned char **) calloc (256,sizeof(unsigned char *)) ;
   for ( j_1=0 ; j_1<256 ; j_1++ )
      {
       tr_SIZ [j_1] = (unsigned char *) calloc (256,sizeof(unsigned char)
       if (tr_SIZ[j_1] == NULL)
         {
          puts (" tr_SIZ alloc ERROR : unsufficent memory . ") ;
          puts (" Program aborted ! ") ;
          exit (1) ;
         }
      }

   tr_ADD = (unsigned int **) calloc (256,sizeof(unsigned int *)) ;
   for ( j_2=0 ; j_2<256 ; j_2++ )
      {
       tr_ADD [j_2] = (unsigned int *) calloc (256,sizeof(unsigned int))
       if (tr_ADD[j_1] == NULL)
         {
          puts (" tr_ADD alloc ERROR : unsufficent memory . ") ;
          puts (" Program aborted ! ") ;
          exit (1) ;
         }
      }

   orga = (unsigned int *) malloc (nbr * sizeof(unsigned int)) ;
   if (orga == NULL)
     {
      puts (" orga alloc ERROR : unsufficient memory . ") ;
      puts (" Program aborted ! ") ;
      exit (1) ;
     }

   for ( j_3=0 ; j_3<nbr ; j_3++ )
      {
       unsigned int val_1 , val_2 ;

       val_1 = *(*(base + j_3)) ;
       val_2 = *((*(base + j_3))+1) ;
       tr_SIZ [val_1][val_2] ++ ;

       orga [j_3] = j_3 ;
      }

   qsort ( orga , nbr , sizeof(unsigned int) , no_cmp ) ;

   offset = 0 ;

   for ( j_4=0; j_4<256; j_4++)
   for ( j_5=0; j_5<256; j_5++)
      {
       if ( tr_SIZ [j_4][j_5] != 0 )
         {
          tr_ADD [j_4][j_5] = offset ;
          offset = offset + tr_SIZ [j_4][j_5] ;
         }
       else tr_ADD [j_4][j_5] = 0 ;
      }

   P_e = clock () ;
/* end pre_pro */
```

```
beg = clock () ;
for ( remake=0 ; remake<stop ; remake++ )
  {

/* begin meth */

   tot = 0 ;

   cc_1 = *(c_cube) ;
   cc_2 = *(c_cube + 1) ;

   ht = min ((cc_2 + m_cube) , 255) ;
   bs = max ( 0 , (cc_2 - m_cube) ) ;
   gh = max ( 0 , (cc_1 - m_cube) ) ;
   dt = min ((cc_1 + m_cube) , 255) ;

   for (I=gh ; I<=dt ; I++)
   for (J=bs ; J<=ht ; J++)
      {
       if ( tr_SIZ [I][J] != 0 )
         {
          unsigned int sub = 0 ;

          unsigned int IJ_siz ;
          unsigned int j_6 ;

          IJ_siz = tr_SIZ [I][J] ;

          for (j_6=0 ; j_6<IJ_siz ; j_6++)
             {
              unsigned char bool = 1 ;
              unsigned int  K = 2 ;

              while ( bool && (K<dim))
                    {
                     unsigned int  IJ_add ;
                     unsigned char cc_K , pt_K ;

                     IJ_add = (tr_ADD [I][J] + j_6) ;
                     cc_K = *(c_cube + K) ;
                     pt_K = *((*(base + (orga[IJ_add])))+K) ;

                     if (((cc_K-m_cube) > pt_K ) || ((cc_K+m_cube) < pt_K )
                        bool = 0 ;

                     K ++ ;
                    }
               sub = sub + bool ;
             }
           tot = tot + sub ;
         }
       }

 /* end meth */

   }
 end = clock () ;

    res = tot ;

    free (tr_ADD) ;
    free (tr_SIZ) ;
    free ( orga ) ;
  }

 /* end proj_meth */
```

```
/* begin ecr_res */

void ecr_res ( )

  {
   unsigned int  i_4 ;

   unsigned char trace = 0 ;

   printf ("\n\nCES.RES >>> \n\nDim. de l' espace : %u\n\n" , dim ) ;

   printf ("Training_Set : \n") ;

   printf ("  Nom du fichier_base < %s :\n" , fb_nom ) ;
   printf ("  Nombre d' elements : %u\n\n" , nbr ) ;

   printf ("Hyper_Cube : \n") ;

   printf ("  Mesure (H) : %u\n" , m_cube ) ;
   printf ("  Centre (Point a classer!) <" ) ;
   for (i_4 = 0 ; i_4<(dim-1) ; i_4++)
       printf (" %u" , *(c_cube+i_4) ) ;
   printf (" %u\n\n" , *(c_cube+(dim-1)) ) ;

   if (altern)
      printf ("By natural_method ! \n") ;
   else
     {
      printf ("By projective_method ! \n") ;
      printf ("\nTime for pre_pro : %f sec.\n"
               , (P_e - P_b) / CLK_TCK ) ;
     }

   printf ("\nSilv_Estimateur => %u\n" , res ) ;

   printf ("\nChrono (1/%u sec / CES) : %f\n"
           , stop , (end - beg) / CLK_TCK ) ;

   printf ("\n<<< CES.RES\n\n") ;
```

```
   printf ("Trace ? [ type 1 (0) and return for Yes (No) ] \n") ;
   scanf ("%u" , &trace ) ;

   if (trace)
     {
      FILE          *f_trace ;
      char          ft_nom[80] ;
      unsigned int i_5 ;

      printf ("\nNom du fichier trace ? \n") ;
      scanf ("%s" , &ft_nom ) ;

      f_trace = fopen (ft_nom , "a+t") ;

      if (f_trace == NULL)
        {
         puts (" fopen ERROR : protected or ... file . ") ;
         puts (" Program aborted ! ") ;
         exit (1) ;
        }

      fprintf (f_trace ,
               "\n CES >>> \n\n Dim. de l' espace : %u\n\n" , dim ) ;

      fprintf (f_trace , " Training_Set : \n") ;

      fprintf (f_trace , "   Nom du fichier_base < %s :\n" , fb_nom ) ;
      fprintf (f_trace , "   Nombre d' elements : %u\n\n" , nbr ) ;

      fprintf (f_trace , " Hyper_Cube : \n") ;

      fprintf (f_trace , "   Mesure (H) : %u\n" , m_cube ) ;
      fprintf (f_trace , "   Centre (Point a classer!) <" ) ;
      for (i_5 = 0 ; i_5<(dim-1) ; i_5++)
          fprintf (f_trace , " %u" , *(c_cube+i_5) ) ;
      fprintf (f_trace , " %u\n\n" , *(c_cube+(dim-1)) ) ;

      if (altern)
         fprintf (f_trace , " By natural_method ! \n") ;
      else
        {
         fprintf (f_trace , "By projective_method ! \n") ;
         fprintf (f_trace , "\nTime for pre_pro : %f sec. \n"
                          , (P_e - P_b) / CLK_TCK ) ;
        }

      fprintf (f_trace , " Silv_Estimateur => %u\n\n" , res ) ;

      fprintf (f_trace , "\nChrono (1/%u sec / CES) : %f\n"
               , stop , (end - beg) / CLK_TCK ) ;

      fclose ( f_trace ) ;
     }

    clrscr () ;
   }

/* end ecr_res */
```

```
/* begin clot_prog */

void clot_prog ( )

  {
   printf ("\n The program CES 's conceived by E.BOURGEOIS ... ") ;
   printf ("\n Implemented on (Borland) Turbo C 2.0 in 1992 for :\n") ;

   printf ("\n ""Mesure de densite(s) en Analyse Discriminante :"" ") ;
   printf ("\n ""Algorithmes , Programmation C et") ;
   printf (" Comparaison de Methodes.""\n") ;
   printf ("\n Memoire presente pour l' obtention du grade de") ;
   printf (" Licencie en Sciences .\n") ;

   printf ("\n Promoteur : Professeur J.P. RASSON .\n") ;

   printf ("\n Facultes Universitaires de Namur . ") ;
   printf ("\n Departement de Mathematique . \n\n") ;

   puts (" Press any key for cloture ! ") ;
   getch () ;
   clrscr () ;

   free ( base ) ;
   free (c_cube) ;

   exit (0) ;
  }

 /* end clot_prog */
```

```
/* begin no_cmp */

no_cmp ( p1 , p2 )

unsigned int *p1 , *p2 ;

  {
   if  ( *(*(base + *p1)) == *(*(base + *p2)) )
      return ( *((*(base + *p1))+1) - *((*(base + *p2))+1) ) ;
   else
      return ( *(*(base + *p1)) - *(*(base + *p2)) ) ;
  }

/* end no_cmp */
```

```
/* begin elt_cmp */

elt_cmp ( e1 , e2 )

sa_elt *e1 , *e2 ;

  {
   if ((e1 -> val) == (e2 -> val))
      return ((e1 -> no) - (e2 -> no)) ;
   else
      return ((e1 -> val) - (e2 -> val)) ;
  }

/* end elt_cmp */
```

```
/* begin ai_excess */

unsigned int ai_excess ( key , pt_sa , nelem )

unsigned char key ;
sa_elt        *pt_sa ;
unsigned int  nelem ;

  {
   unsigned int n_1 ;
   int          aux_1 ;

   n_1 = 0 ;

   do {
       aux_1 = b_s ( key + n_1 , pt_sa , nelem ) ;

       n_1 = n_1 + 1 ;
      }
   while ( aux_1 == -1 ) ;

   while ((aux_1 == 0)||(pt_sa[aux_1-1].val == pt_sa[aux_1].val))
        {
         if (aux_1 == 0)
            return (aux_1) ;
         else
            aux_1 = aux_1 - 1 ;
        }

   return (aux_1) ;

  }

/* end ai_excess */
```

```
/* begin ai_default */

unsigned int ai_default ( key , pt_sa , nelem )

unsigned char key ;
sa_elt        *pt_sa ;
unsigned int  nelem ;

  {
   unsigned int n_2 ;
   int          aux_2 ;

   n_2 = 0 ;

   do {
       aux_2 = b_s ( key - n_2 , pt_sa , nelem ) ;

       n_2 = n_2 + 1 ;
      }
   while ( aux_2 == -1 ) ;

   while ((aux_2 == (nelem - 1))||(pt_sa[aux_2+1].val == pt_sa[aux_2].val)
        {
         if (aux_2 == (nelem - 1))
            return (aux_2) ;
         else
            aux_2 = aux_2 + 1 ;
        }

   return (aux_2) ;

  }

 /* end ai_default */
```

```
/* begin b_s */

int b_s ( key , pt_sa , nelem )

unsigned char key ;
sa_elt        *pt_sa ;
unsigned int  nelem ;

  {
   unsigned int low_i , high_i , mid_i ;

   low_i = 0 ;

   high_i = nelem - 1 ;

   while (low_i <= high_i)
        {
         mid_i = (low_i + high_i) / 2 ;

         if ( key < pt_sa[mid_i].val )
            high_i = mid_i - 1 ;

         else if ( key > pt_sa[mid_i].val )
                 low_i = mid_i + 1 ;

              else return ( mid_i ) ;
        }

   return (-1) ;
  }

/* end b_s */


/* END CES.C */
```